KQML as an Agent Communication Language
(1994) (Make Corrections) (382 citations)
Tim Finin, Richard Fritzson, Don McKay, Robin McEntire
Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)

Abstract: This paper describes the design of and experimentation with the Knowledge Query and Manipulation Language (KQML), a new language and protocol for exchanging information and knowledge. This work is part of a larger effort, the ARPA Knowledge Sharing Effort which is aimed at developing techniques and methodology for building large-scale knowledge bases which are sharable and reusable. KQML is both a message format and a message-handling protocol to support run-time knowledge sharing among agents. ... (Update)

Context of citations to this paper: More

...ndcessaires h la communication entre les agents en plus de fournir un vdrificateur de la syntaxe de KQML. **Rappelons que KQML [14] est un langage d interrogation et de manipulation des connaissances.** C est un langage basd sur les acres du langage naturel [8] I1...

...and it s modification in an explicit way. **All the agents use Knowledge Query and Manipulation Language (KQML) for communication [8].** AP and PA multiagent system play a key role in enabling INTERLABS distributed virtual campus Each INTERLABS client uses a Web navigator...

Cited by: More
Engineering Infrastructures for Mobile Organizations - Cabri, Leonardi, Mamei.. (2001) (Correct)
Interaction Modal Logic for multiagent systems based - On Bdi Architecture (Correct)
Stream Oriented Interactions for Highly - Distributed And Disconnected (Correct)

Similar documents (at the sentence level): More
31.0%: KQML as an Agent Communication Language - Finin, Labrou, Mayfield (1994) (Correct)
27.4%: KQML as an agent communication language - Finin, Labrou, Mayfield (1995) (Correct)
19.1%: Desiderata for Agent Communication Languages - Mayfield, Labrou, Finin (1995) (Correct)

Active bibliography (related documents): More All
0.3: Cognitive Modeling and Group Adaptation in Intelligent.. - Leonardo Garrido (Correct)
0.3: Dynamic Agency: a Methodology and Architecture for Multiagent.. - Amigoni (2000) (Correct)
0.2: A Security Architecture for Agent Communication Languages - Mayfield, Finin (Correct)

Similar documents based on text: More All
0.8: An Overview of KQML: A Knowledge Query and.. - Chalupsky, Finin.. (1992) (Correct)
0.7: A Security Architecture Based on Trust Management for.. - Systems Lalana Kagal (2002) (Correct)
0.7: A Reactive Service Composition Architecture for.. - Chakraborty.. (2002) (Correct)

KQML as an Agent Communication Language *

Tim Finin and Richard Fritzson
Computer Science Department
University of Maryland Baltimore County
Baltimore MD USA
finin@cs.umbc.edu
fritzson@cs.umbc.edu

Don McKay and Robin McEntire
Valley Forge Laboratory
Unisys Corporation
Paoli PA USA
mckay@vfl.paramax.com
robin@vfl.paramax.com

Abstract

This paper describes the design of and experimentation with the Knowledge Query and Manipulation Language (KQML), a new language and protocol for exchanging information and knowledge. This work is part of a larger effort, the ARPA Knowledge Sharing Effort which is aimed at developing techniques and methodology for building large-scale knowledge bases which are sharable and reusable. KQML is both a message format and a message-handling protocol to support run-time knowledge sharing among agents. KQML focuses on an extensible set of *performatives*, which defines the permissible "speech acts" agents may use and comprise a substrate on which to develop higher-level models of interagent interaction such as contract nets and negotiation. In addition, KQML provides a basic architecture for knowledge sharing through a special class of agent called *communication facilitators* which coordinate the interactions of other agents The ideas which underlie the evolving design of KQML are currently being explored through experimental prototype systems which are being used to support several testbeds in such areas as concurrent engineering, intelligent design and intelligent planning and scheduling.

1 Introduction

The computational environment which is emerging in such programs as the National Information Infrastructure (NII) is characterized by being highly distributed, heterogeneous, extremely dynamic, and comprising a large number of autonomous nodes. An information system operating in such an environment must handle several emerging problems:

- The predominant architecture on the Internet, the client-server model, is too restrictive. It is difficult for current Internet information services to take the initiative in bringing new, critical material to a user's attention. Some nodes will want to act as both clients and servers, depending on who they are interacting with.
- Several forms of heterogeneity need to be handled, e.g. different platforms, different data formats, the capabilities of different information services, and the implementation technologies employed.
- Many software technologies such as event simulation, applied natural language processing, knowledge–based reasoning, advanced information retrieval, speech processing, etc. have matured to the point of being ready to participate in and contribute to an NII type environment. However, there is a lack of tools and techniques for constructing intelligent clients and servers or for building agent–based software in general.

A community of *intelligent agents* can address each of the problems mentioned above. When we describe these agents as intelligent, we refer to their ability to: communicate with each other using an expressive communication language; work together cooperatively to accomplish complex goals; act on their own initiative; and use local information and knowledge to manage local resources and handle requests from peer agents.

Knowledge Query and Manipulation Language (KQML) is a language that is designed to support interactions among intelligent software agents. It was developed by the ARPA supported Knowledge Sharing Effort [24, 27] and separately implemented by several research groups. It has been successfully used to implement a variety of information systems using different software architectures.

The Knowledge Sharing Effort

The ARPA Knowledge Sharing Effort (KSE) is a consortium to develop conventions facilitating sharing and reuse of knowledge bases and knowledge based systems. Its goal is to define, develop, and test infrastructure and supporting technology to enable participants to build much bigger and more broadly functional systems than could be achieved working alone. The KSE is organized around four working groups each of which addresses a complementary problem identified in current knowledge representation technology: *Interlingua*, *KRSS*, *SRKB*, and *External Interfaces*.

The *Interlingua Group* is developing a common language for expressing the content of a knowledge-base. This group has published a specification document describing the *Knowledge Interchange Formalism* or *KIF* [15] which is based on first order logic with some extensions to support non-monotonic reason and definitions. KIF includes both a specifica-

*This work was supported in part by the Air Force Office of Scientific Research under contract F49620-92-J-0174, and the Advanced Research Projects Agency monitored under USAF contracts F30602-93-C-0177 and F30602-93-C-0028 by Rome Laboratory.

To appear in *The Proceedings of the Third International Conference on Information and Knowledge Management* (CIKM'94), ACM Press, November 1994.

tion of a syntax for the language as well as a specification for the semantics. KIF can be used to support the translation from one content language to another or as a common content language between two agents which use different native representation languages. Information of KIF and associated tools and is available from `http://www.cs.umbc.edu/kse/kif/` .

The *KRSS Group* (Knowledge Representation System Specification) is focussed on defining common constructs within families of representation languages. It has recently finished a common specification for terminological representations in the KL-ONE family. This document and other information on the KRSS group is available as `http://www.cs.umbc.edu/kse/krss/` .

The *SRKB Group* (Shared, Reusable Knowledge Bases) is concerned with facilitating consensus on contents of sharable knowledge bases, with sub-interests in shared knowledge for particular topic areas and in topic-independent development tools and methodologies. It has established a repository for sharable ontologies and tools which is available over the Internet as `http://www.cs.umbc.edu/kse/srkb/` .

The scope of the *External Interfaces Group* is the run-time interactions between knowledge based systems and other modules in a run-time environment. Special attention has been given to two important cases – communication between two knowledge-based systems and communication between a knowledge-based system and a conventional database management system [26]. The KQML language is one of the main results which have come out of the external interfaces group of the KSE. General information is available from `http://www.cs.umbc.edu/kqml`.

2 KQML and Intelligent Information Integration

We could address many of the difficulties of communication between intelligent agents described in the Introduction by giving them a common language. In linguistic terms, this means that they would share a common syntax, semantics and pragmatics.

Getting information processes, especially AI processes, to share a common syntax is a major problem. There is no universally accepted language in which to represent information and queries. Languages such as KIF [15], extended SQL, and LOOM [22] have their supporters, but there is also a strong position that it is too early to standardize on any representation language [19]. As a result, it is currently necessary to say that two agents can communicate with each other if they have a common representation language or use languages that are inter-translatable.

Assuming a common or translatable language, it is still necessary for communicating agents to share a framework of knowledge in order to interpret message they exchange. This is not really a shared semantics, but a shared ontology. There is not likely to be one shared ontology, but many. Shared ontologies are under development in many important application domains such as planning and scheduling, biology and medicine.

Pragmatics among computer processes includes 1) knowing who to talk with and how to find them and 2) knowing how to initiate and maintain an exchange. KQML is concerned primarily with pragmatics (and secondarily with semantics). It is a language and a set of protocols that support computer programs in identifying, connecting with and exchanging information with other programs.

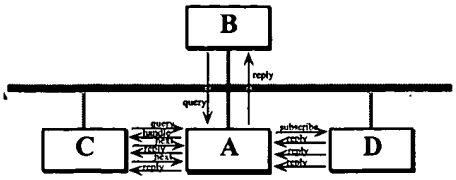


Figure 1: **Several basic communication protocols are supported in KQML.**

Agent Communication Protocols

There are a variety of interprocess information exchange protocols. In the simplest, one agent acts as a client and sends a query to another agent acting as a server and then waits for a reply, as is shown between agents A and B in Figure 1. The server's reply might consist of a single answer or a collection or set of answers. In another common case, shown between agents A and C, the server's reply is not the complete answer but a handle which allows the client to ask for the components of the reply, one at a time. A common example of this exchange occurs when a client queries a relational database or a reasoner which produces a sequence of instantiations in response. Although this exchange requires that the server maintain some internal state, the individual transactions are as before – involving a *synchronous* communication between the agents. A somewhat different case occurs when the client subscribes to a server's output and an indefinite number of *asynchronous* replies arrive at irregular intervals, as between agents A and D in Figure 1. The client does not know when each reply message will be arriving and may be busy performing some other task when they do.

There are other variations of these protocols. Messages might not be addressed to specific hosts, but broadcast to a number of them. The replies, arriving synchronously or asynchronously have to be collated and, optionally, associated with the query that they are replying to.

Facilitators and Mediators

One of the design criteria for KQML was to produce a language that could support a wide variety of interesting agent architectures. Our approach to this is to introduce a small number of KQML performatives which are used by agents to describe the meta-data specifying the information requirements and capabilities and then to introduce a special class of agents called *communication facilitators* [16]. A facilitator is an agent that performs various useful communication services, e.g. maintaining a registry of service names, forwarding messages to named services, routing messages based on content, providing "matchmaking" between information providers and clients, and providing mediation and translation services.

As an example, consider a case where an agent A would like to know the truth of a sentence X, and agent B may have X in its knowledge-base, and a facilitator agent F is available. If A is aware that it is appropriate to send a query about X to B, then it can use a simple *point to point* protocol and send the query directly to B, as in Figure 2. If, however, A is not aware of what agents are available, or which may have X in their knowledge-bases, or how to contact those of whom it is aware, then a variety of approaches can be used. Figure 3 shows an example in which A uses the *subscribe* performative to request that facilitator F monitor for the truth of X. If B subsequently informs F that it believes X to be true, then F can in turn inform A.

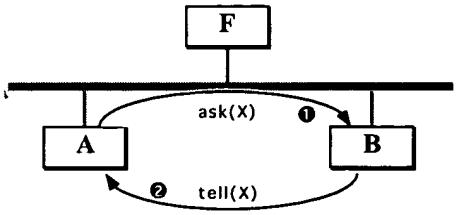


Figure 2: **When A is aware of B and of the appropriateness of querying B about X, a simple point-to-point protocol can be used.**

Figure 4 shows a slightly different situation. A asks F to find an agent that can process an *ask(X)* performative. B independently informs F that it is willing to accept performatives matching *ask(X)*. Once F has both of these messages, it sends B the query, gets a response and forwards it to A.

In Figure 5, A uses a slightly different performative to inform F of its interest in knowing the truth of X. The recruit performative asks the recipient to find an agent that is willing to receive and process an embedded performative. That agent's response is then to be directly sent to the initiating agent. Although the difference between the examples used in Figures 3 and 5 are small for a simple ask query, consider what would happen if the embedded performative was *subscribe(ask-all(X))*.

As a final example, consider the exchange in Figure 6 in which A asks F to "recommend" an agent to whom it would be appropriate to send the performative *ask(X)*. Once F learns that B is willing to accept *ask(X)* performatives, it replies to A with the name of agent B. A is then free to initiate a dialog with B to answer this and similar queries.

From these examples, we can see that one of the main functions of facilitator agents is to help other agents find appropriate clients and servers. The problem of how agents find facilitators in the first place is not strictly an issue for KQML and has a variety of possible solutions.

Current KQML-based applications have used one of two simple techniques. In the PACT project [7], for example, all agents used a central, common facilitator whose location was a parameter initialized when the agents were launched. In the ARPI applications [5], finding and establishing contact with a local facilitator is one of the functions of the KQML API. When each agent starts up, its KQML router module announces itself to the local facilitator so that it is registered in the local database. When the application exits, the router sends another KQML message to the facilitator, removing the application from the facilitator's database. By

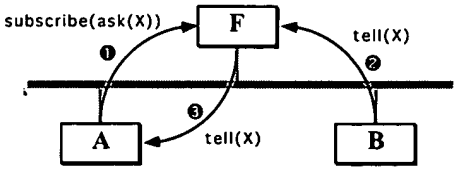


Figure 3: **Agent A can ask facilitator agent F to monitor for changes in its knowledge-base. Facilitators are agents that deal in knowledge about the information services and requirements of other agents and offer such services as forwarding, brokering, recruiting and content-based routing.**

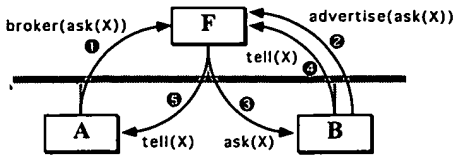


Figure 4: **The broker performative is used to ask a facilitator agent to find another agent which can process a given performative and to receive and forward the reply.**

convention, a facilitator agent should be running on a host machine with the symbolic address *facilitator.domain* and listening to the standard KQML port.

Scaling up to a national-scale information enterprise will require the incorporation of new techniques. The current Internet *Domain Name Servers* (DNS) use a very simple, yet robust technique for mapping symbolic names into internet IP addresses. Similar techniques can be used to map symbolic agent "names" into specific agent references that can be used to contact the agent. What will be involved is the development of a hierarchical "ontology" for organizing information that is orthogonal to the hierarchical scheme used to organize the Internet. Figure 7 shows such an agent which could function as such facilitator-agent-server.

The role of KQML

As a communication language for intelligent information agents, KQML draws on work in both *distributed systems* and *distributed AI* and offers a level of abstraction that should be useful to both.

With respect to distributed software systems in general, KQML provides an abstraction of a process as an information agent as a knowledge-based system (KBS). The KBS model easily subsumes a broad range of commonly used information agent models in use today, including database management systems, hypertext systems, server-oriented software (e.g. finger demons, mail servers, HTML servers, etc), simulations, etc. Such systems can usually be modeled as having two virtual knowledge bases – one representing the agent's information store (i.e., beliefs) and the other representing its intentions (i.e., goals). We hope that future standards for interchange and interoperability languages and protocols will be based on this very powerful and rich model. This will avoid the built-in limitations of more constrained models (e.g., that of a simple remote procedure call or relational database query) and also make it easier to integrate truly intelligent agents with simpler and more mundane information clients and servers. Current KQML implementations have used standard communication and messaging protocols as a transport layer, including TCP/IP, email, Linda, HTTP, and CORBA. As standards in this area evolve and

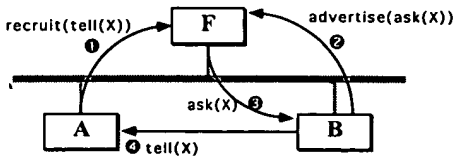


Figure 5: **The recruit performative is used to ask a facilitator agent to find an appropriate agent to which an embedded performative can be forwarded. Any reply is returned directly to the original agent.**

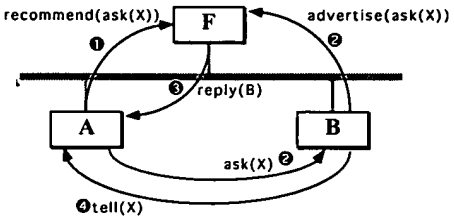


Figure 6: **The recommend performative is used to ask a facilitator agent to respond with the "name" of another agent which is appropriate for sending a particular performative.**

new standards are introduced, we expect that KQML implementations will use them.

The contribution that KQML makes to Distributed AI research is to offer a standard language and protocol that intelligent agents can use to communicate among themselves as well as with other information servers and clients. The independence of the communication and content languages affords a flexibility which is quite useful. In designing KQML, our goal is to build in the primitives necessary to support all of the interesting agent architectures currently in use. If we have been successful, then KQML should serve to be a good tool for DAI research, and, if used widely, should enable greater research collaboration among DAI researchers.

3 The KQML Language

Communication takes place on several levels. The content of the message is only a part of the communication. Being able to locate and engage the attention of someone you want to communicate with is a part of the process. Packaging your message in a way which makes your purpose in communicating clear is another.

When using KQML, a software agent transmits content messages, composed in a language of its own choice, wrapped inside of a KQML message. The content message can be expressed in any representation language and written in either ASCII strings or one of many binary notations (e.g. network independent XDR representations). All KQML implementations ignore the content portion of the message except to the extent that they need to recognize where it begins and ends.

The syntax of KQML is based on a balanced parenthesis list. The initial element of the list is the performative and the remaining elements are the performative's arguments as keyword/value pairs. Because the language is relatively simple, the actual syntax is not significant and can be changed if necessary in the future. The syntax reveals the roots of the initial implementations, which were done in Common Lisp, but has turned out to be quite flexible.

KQML is expected to be supported by an software substrate which makes it possible for agents to locate one another in a distributed environment. Most current implementations come with custom environments of this type; these are commonly based on helper programs called *routers* or *facilitators*. These environments are not a specified part of KQML. They are not standardized and most of the current KQML environments will evolve to use some of the emerging commercial frameworks, such as OMG's CORBA or Microsoft's OLE2, as they become more widely used.

The KQML language supports these implementations by allowing the KQML messages to carry information which is

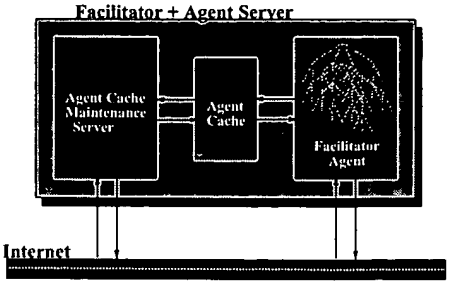


Figure 7: **Some facilitator agents will specialize in knowing how to contact other agents (among other things) and can thus act as "agent-servers".**

useful to them, such as the names and addresses of the sending and receiving agents, a unique message identifier, and notations by any intervening agents. There are also optional features of the KQML language which contain descriptions of the content: its language, the ontology it assumes, and some type of more general description, such as a descriptor naming a topic within the ontology. These optional features make it possible for the supporting environments to analyze, route and deliver messages based on their content, *even though the content itself is inaccessible.*

The forms of these parts of the KQML message may vary, depending on the transport mechanism used to carry the KQML messages. In implementations which use TCP streams as the transport mechanism, they appear as fields in the body of the message. In an earlier version of KQML, these fields were kept in *reserved* locations, in an outer wrapper of the message, to emphasize the difference from other fields. In other transport mechanisms the syntax and content of these message may be different. For example, in the E-mail implementation of KQML, these fields are embedded in KQML mail headers.

The set of performatives forms the core of the language. It determines the kinds of interactions one can have with a KQML-speaking agent. The primary function of the performatives is to identify the protocol to be used to deliver the message and to supply a *speech act* which the sender attaches to the content. The performative signifies that the content is an *assertion*, a *query*, a *command*, or any other mutually agreed upon speech act. It also describes how the sender would like any reply to be delivered, that is, what protocol will be followed.

Conceptually, a KQML message consists of a performative, its associated arguments which include the real content of the message, and a set of optional arguments *transport* which describe the content and perhaps the sender and receiver. For example, a message representing a query about the price of a share of IBM stock might be encoded as:

```
(ask-one
  :content (PRICE IBM ?price)
  :receiver stock-server
  :language LPROLOG
  :ontology NYSE-TICKS)
```

In this message, the KQML performative is *ask-one*, the content is *(price ibm ?price)*, the ontology assumed by the query is identified by the token *nyse-ticks*, the receiver of the message is to be a server identified as *stock-server* and the query is written in a language called *LPROLOG*. A similar query could be conveyed using standard Prolog as the con-

tent language in a form that requests the set of all answers as:

```
(ask-all
  :content "price(IBM, [?price, ?time])"
  :receiver stock-server
  :language standard_prolog
  :ontology NYSE-TICKS)
```

The first message asks for a single reply; the second asks for a set as a reply. If we had posed a query which had a large number of replies, would could ask that they each be sent separately, instead of as a single large collection by changing the performative. (To save space, we will no longer repeat fields which are the same as in the above examples.)

```
(stream-all
  ;;?VL is a large set of symbols
  :content (PRICE ?VL ?price))
```

The *stream-all* performative asks that a set of answers be turned into a set of replies. To exert control of this set of reply messages we can wrap another performative around the preceding message.

```
(standby
   :content (stream-all
              :content (PRICE ?VL ?price)))
```

The *standby* performative expects a KQML language content and it requests that the agent receiving the request take the stream of messages and hold them and release them one at a time, each time the sending agent transmits a message with the *next* performative. The exchange of next/reply messages can continue until the stream is depleted or until the sending agent sends either a *discard* message (i.e. discard all remaining replies) or a *rest* message (i.e. send all of the remaining replies now). This combination is so useful that it can be abbreviated:

```
(generate
    :content (PRICE ?VL ?price)))
```

A different set of answers to the same query can be obtained (from a suitable server) with the query:

```
(subscribe
   :content (stream-all
              :content (PRICE IBM ?price)))
```

This performative requests all future changes to the answer to the query, i.e. it is a stream of messages which are generated as the trading price of IBM stock changes. An abbreviation for subscribe/stream combination is known a *monitor*.

```
(monitor
    :content (PRICE IBM ?price)))
```

Though there is a predefined set of reserved performatives, it is neither a minimal required set nor a closed one. A KQML agent may choose to handle only a few (perhaps one or two) performatives. The set is extensible; a community of agents may choose to use additional performatives if they agree on their interpretation and the protocol associated with each. However, an implementation that chooses to implement one of the reserved performatives must implement it in the standard way.

Basic query performatives:

- evaluate, ask-if, ask-in, ask-one, ask-all, ...

Multi-response query performatives:

- stream-in, stream-all, ...

Response performatives:

- reply, sorry, ...

Generic informational performatives:

- tell, achieve, cancel, untell, unachieve, ...

Generator performatives:

- standby, ready, next, rest, discard, generator, ...

Capability-definition performatives:

- advertise, subscribe, monitor, import, export, ...

Networking performatives:

- register, unregister, forward, broadcast, route, ...

Figure 8: **There are about two dozen reserved performative names which fall into seven basic categories.**

Some of the reserved performatives are shown in Figure 8. In addition to standard communication performatives such as ask, tell, deny, delete, and more protocol oriented performatives such as *subscribe*, KQML contains performatives related to the non-protocol aspects of pragmatics, such as *advertise* - which allows an agent to announce what kinds of asynchronous messages it is willing to handle; and *recruit* - which can be used to find suitable agents for particular types of messages. For example, the server in the above example might have earlier announced:

```
(advertise
    :ontology NYSE-TICKS
    :language LPROLOG
    :content (monitor
               :content (PRICE ?x ?y)))
```

Which is roughly equivalent to announcing that it is a stock ticker and inviting monitor requests concerning stock prices. This *advertise* message is what justifies the subscriber's sending the *monitor* message.

4 KQML Software Architectures

KQML was not defined by a single research group for a particular project. It was created by a committee of representatives from different projects, all of which were concerned with managing distributed implementations of systems. One was a distributed collaboration of expert systems in the planning and scheduling domain. Another was concerned with problem decomposition and distribution in the CAD/CAM domain. A common concern was the management of a collection of cooperating processes and the simplification of the programming requirements for implementing a system of this type. However, the groups did not share a common communication architecture. As a result, KQML does not dictate a particular system architecture, and several different systems have evolved.

Our group has two implementations of KQML. One is written in Common Lisp, the other in C. Both are fully interoperable and are frequently used together. The design of these implementations was motivated by the need to integrate a variety of preexisting expert systems into a collaborating group of processes. Most of the systems involved were never designed to operate in a communication oriented

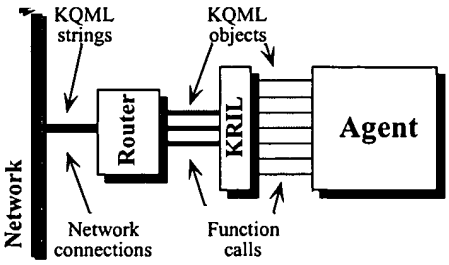


Figure 9: **A router gives an application a single interface to the network, providing both client and server capabilities, managing multiple simultaneous connections, and handling some KQML interactions autonomously. The KRIL is the interface to the agent application and provides internal access points to which the router delivers incoming messages, analyzes outgoing messages for appropriate domain tagging and routing, and provides application specific interface and procedures for communication access.**

environment. The design is built around two specialized programs, a *router* and a *facilitator*, and a library of interface routines, called a *KRIL*.

KQML Routers

Routers are content independent message routers. Each KQML speaking software agent is associated with its own separate router process. All routers are identical; each is just an executing copy of the same program. A router handles all KQML messages going to and from its associated agent. Because each program has an associated router process, it is not necessary to make extensive changes to each program's internal organization to allow it to asynchronously receive messages from a variety of independent sources. The router provides this service for the agent and provides the agent with a single point of contact for the rest of the network. It provides both client and server functions for the application and manages multiple simultaneous connections with other agents.

The router never looks at the content fields of the messages it handles. It relies on the KQML performatives and its arguments. If an outgoing KQML message specifies a particular Internet address, the router directs the message to it. If the message specifies a particular service, the router will attempt to find an Internet address for that service and deliver the message to it. If the message only provides a description of the content (e.g. query, :ontology "geo-domain-3", :language "Prolog", etc.) the router may attempt to find a server which can deal with the message and it will deliver it there, or it may choose to forward it to a smarter communication agent which may be willing to route it. Routers can be implemented with varying degrees of sophistication – they can not guarantee to deliver all messages.

KQML Facilitators

To deliver messages that are incompletely addressed, routers rely on *facilitators*. A facilitator is a network application which provides useful network services. It maintains a registry of service names; it will forward messages on request to named services. It may provide matchmaking services between information providers and consumers. Facilitators are actual network software agents which have their own KQML routers to handle their traffic and deal exclusively in KQML messages. There is typically one facilitator for each local group of agents. This can translate into one facilitator per local site or one per project; there may be multiple local facilitators to provide redundancy. When each application starts up, its router announces itself to the local facilitator so that it is registered in the local database. When the application exits, the router sends another KQML message to the facilitator, removing the application from the facilitator's database. In this way applications can find each other without there having to be a hand maintained list of local services.

KQML KRILs

Since the router is a separate process from the application, it is necessary to have a programming interface between the application and the router. This application program interface (API) is called a KRIL (KQML Router Interface Library). While the router is a separate process, with no understanding of the content field of the KQML message, the KRIL API is embedded in the application and has access to the application's tools for analyzing the content. While there is only one piece of router code, which is instantiated for each process, there can be various KRILs, one for each application type and one for each application language. The general goal of the KRIL is to make access to the router as simple as possible for the programmer.

To this end, a KRIL can be as tightly embedded in the application, or even the application's programming language, as is desirable. For example, an early implementation of KQML featured a KRIL for the Prolog language which had only a simple declarative interface for the programmer. During the operation of the Prolog interpreter, whenever the Prolog database was searched for predicates, the KRIL would intercept the search; determine if the desired predicates were actually being supplied by a remote agent; formulate and pose an appropriate KQML query; and return the replies to the Prolog interpreter as though they were recovered from the internal database. The Prolog program itself contained no mention of the distributed processing going on except for the declaration of which predicates were to be treated as remote predicates.

It is not necessary to completely embed the KRIL in the application's programming language. A simple KRIL generally provides two programmatic entries. For initiating a transaction there is a **send-kqml-message** function. This accepts a message content and as much information about the message and its destination as can be provided and returns either the remote agent's reply (if the message transmission is synchronous and the process blocks until a reply is received) or a simple code signifying the message was sent. For handling incoming asynchronous messages, there is usually a **declare-message-handler** function. This allows the application programmer to declare which functions should be invoked when messages arrive. Depending on the KRILs capabilities, the incoming messages can be sorted according to *performative*, or *topic*, or other features, and routed to different message handling functions.

In addition to these programming interfaces, KRILs accept different types of declarations which allow them to register their application with local facilitators and contact remote agents to advise them that they are interested in receiving data from them. Our group has implemented a variety of experimental KRILs, for Common Lisp, C, Prolog, Mosaic, SQL, and other tools.

5 Experience with KQML

The KQML language and implementations of the protocol have been used in several prototype and demonstration systems. The applications have ranged from concurrent design and engineering of hardware and software systems, military transportation logistics planning and scheduling, flexible architectures for large-scale heterogeneous information systems, agent-based software integration and cooperative information access planning and retrieval. KQML has the potential to significantly enhance the capabilities and functionality of large-scale integration and interoperability efforts now underway in communication and information technology such as the national information infrastructure and OMG's CORBA, as well as in application areas electronic commerce, health information systems and virtual enterprise integration. The content languages used have included languages intended for knowledge exchange including the Knowledge Interchange Format (KIF) and the Knowledge Representation Specification Language (KRSL) [21] as well as other more traditional languages such as SQL. Early experimentations with KQML began in 1990. The following is a representative selection of applications and experiments developed using KQML.

The design and engineering of complex computer systems, whether exclusively hardware or software systems or both, today involves multiple design and engineering disciplines. Many such systems are developed in fast cycle or concurrent processes which involve the immediate and continual consideration of end-product constraints, e.g., marketability, manufacturing planning, etc. Further, the design, engineering and manufacturing components are also likely to be distributed across organizational and company boundaries. KQML has proved highly effective in the integration of diverse tools and systems enabling new tool interactions and supporting a high-level communication infrastructure reducing integration cost as well as flexible communication across multiple networking systems. The use of KQML in these demonstrations has allowed the integrators to focus on what the integration of design and engineering tools can accomplish and appropriately deemphasized how the tools communicate [17, 23, 8, 10].

We have used KQML as the communication language in several technology integration experiments in the ARPA Rome Lab Planning Initiative. One of these experiments supported an integrated planning and scheduling system for military transportation logistics linking a planning agent (in SIPE [30, 4]), with a scheduler (in Common Lisp), a knowledge base (in LOOM [22]), and a case based reasoning tool (in Common Lisp). All of the components integrated were preexisting systems which were not designed to work in a cooperative distributed environment.

In a second experiment, we developed a information agent consisting of CoBASE [6], a cooperative front-end, SIMS [1, 2], an information mediator for planning information access, and LIM [26], an information mediator for translating relational data into knowledge structures. CoBASE processes a query, and, if no responses are found relaxes the query based upon approximation operators and domain semantics and executes the query again. CoBASE generates a single knowledge-based query for SIMS which using knowledge of different information sources selects which of several information sources to access, partitions the query and optimizes access. Each of the resulting queries in this experiment is sent to a LIM knowledge server which answers the query by creating objects from tuples in a relational database. A LIM server front-ends each different database. This experiment was run over the internet involving three, geographically dispersed sites.

Agent-Base Software Integration [18] is an effort underway at Stanford University which applying KQML as an integrating framework for general software systems. Using KQML, a federated architecture incorporating a highly sophisticated facilitator is developed which supports an agent-based view of software integration and interoperation [16]. The facilitator in this architecture is an intelligent agent used to process and reason about the content of KQML messages supporting tighter integration of disparate software systems.

We have also successfully used KQML in other smaller demonstrations integrating distributed clients (in C) with mediators which were retrieving data from distributed databases. Mediators are not just limited distributed database access. In another demonstration, we experimented with a KQML URL for the World Wide Web. The static nature of links within such hypermedia structures lends itself to be extended with virtual and dynamic links to arbitrary information sources as can be supported easily with KQML.

6 Conclusion

This paper has described KQML – a language and associated protocol by which intelligent software agents can communicate to share information and knowledge. We believe that KQML, or something very much like it, will be important in building the distributed agent-oriented information systems of the future.

The design of KQML has continued to evolve as the ideas are explored and feedback is received from the prototypes and the attempts to use them in real testbed situations. Furthermore, new standards for sharing persistent object-oriented structures are being developed and promulgated, such as OMG's CORBA specification and Microsoft's OLE 2.0. Should any of these become widely used, it will be worthwhile to evolve KQML so that its key ideas the collection of reserved performatives, the support for a variety of information exchange protocols, the need for an information based directory service can enhance these new information exchange languages.

Additional information on KQML, including papers, language specifications, access to APIs, information on email discussion lists, etc, can be obtained via the world wide web as `http://www.cs.umbc.edu/kqml/` and via ftp from `ftp.cs.umbc.edu` in `pub/kqml/`.

References

[1] Yigal Arens. Planning and reformulating queries for semantically-modeled multidatabase systems. In *First International Conference on Information and Knowledge Management*, October 1992.

[2] Yigal Arens, Chin Chee, Chun-Nan Hsu, Hoh In, and Craig A. Knoblock. Query processing in an information mediator. In *Proceedings of the ARPA/Rome Lab 1994 Knowledge-Based Planning and Scheduling Initiative Workshop*, February 1994.

[3] External Interfaces Working Group ARPA Knowledge Sharing Initiative. Specification of the KQML agent-communication language. Working paper. Available as `http://www.cs.umbc.edu/kqml/papers/kqml-spec.ps`, December 1992.

[4] Marie Bienkowski, Marie desJardins, and Roberto Desimone. SOCAP: system for operations crisis action planning. In *Proceedings of the ARPA/Rome Lab 1994 Knowledge-Based Planning and Scheduling Initiative Workshop*, February 1994.

[5] Mark Burstein, editor. *Proceedings of the ARPA/Rome Lab 1994 Knowledge-Based Planning and Scheduling Initiative Workshop.* Morgan Kuafmann Publishers, Inc., February 1994.

[6] Wes Chu and Hua Yang. Cobase: A cooperative query answering system for database systems. In *Proceedings of the ARPA/Rome Lab 1994 Knowledge-Based Planning and Scheduling Initiative Workshop*, February 1994.

[7] M. Cutkosky, E. Engelmore, R. Fikes, T. Gruber, M. Genesereth, and W. Mark. PACT: An experiment in integrating concurrent engineering systems. *IEEE Computer*, pages 28–38, January 1993.

[8] D. Kuokka et. al. Shade: Technology for knowledge-based collaborative. In *AAAI Workshop on AI in Collaborative Design*, 1993.

[9] J. McGuire et. al. Shade: Technology for knowledge-based collaborative engineering. *Journal of Concurrent Engineering: Applications and Research (CERA)*, 1(2), September 1993.

[10] William Mark et. al. Cosmos: A system for supporting design negotiation. *Journal of Concurrent Engineering: Applications and Research (CERA)*, 2(3), 1994.

[11] Tim Finin, Rich Fritzson, and Don McKay. A high-level language and protocol to support intelligent agent interoperability. In *Workshop on Enabling Technologies for Concurrent Engineering*, April 1992.

[12] Tim Finin, Rich Fritzson, and Don McKay. A knowledge query and manipulation language for intelligent agent interoperability. In *Fourth National Symposium on Concurrent Engineering, CE & CALS Conference*, June 1–4 1992. Available as `http://www.cs.umbc.edu/kqml/papers/cecals.ps`.

[13] Tim Finin, Don McKay, Rich Fritzson, and Robin McEntire. KQML: an information and knowledge exchange protocol. In *International Conference on Building and Sharing of Very Large-Scale Knowledge Bases*, December 1993. A version of this paper will appear in Kazuhiro Fuchi and Toshio Yokoi (Ed.), "Knowledge Building and Knowledge Sharing", Ohmsha and IOS Press, 1994. Available as `http://www.cs.umbc.edu/kqml/papers/kbks.ps`.

[14] Tim Finin, Charles Nicholas, and Yelena Yesha, editors. *Information and Knowledge Management, Expanding the Definition of Database.* Lecture Notes in Computer Science 752. Springer-Verlag, 1993. (ISBN 3-540-57419-0).

[15] M. Genesereth and R. Fikes et. al. Knowledge interchange format, version 3.0 reference manual. Technical report, Computer Science Department, Stanford University, 1992.

[16] Michael R. Genesereth and Steven P. Katchpel. Software agents. *Communications of the ACM*, 37(7):48–53, 147, 1994.

[17] Mike Genesereth. Designworld. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 2,785–2,788. IEEE CS Press.

[18] Mike Genesereth. An agent-based approach to software interoperability. Technical Report Logic-91-6, Logic Group, CSD, Stanford University, February 1993.

[19] Matt Ginsberg. Knowledge interchange format: The KIF of death. *AI Magazine*, 1991.

[20] Yannis Labrou and Tim Finin. A semantics approach for KQML – a general purpose communication language for software agents. In *Third International Conference on Information and Knowledge Management*, November 1994. Available as `http://www.cs.umbc.edu/kqml/papers/kqml-semantics.ps`.

[21] Nancy Lehrer. The knowledge representation specification language manual. Technical report, ISX Corporation, Thousand Oaks, California, 1994.

[22] Robert MacGregor and Raymond Bates. The LOOM knowledge representation language. Technical Report ISI/RS-87-188, USC/ISI, 1987. Also appears in *Proceedings of the Knowledge-Based Systems Workshop* held in St. Louis, Missouri, April 21–23, 1987.

[23] M.Tenenbaum, J. Weber, and T. Gruber. Enterprise integration: Lessons from shade and pact. In C. Petrie, editor, *Enterprise Integration Modeling.* MIT Press, 1993.

[24] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36–56, Fall 1991.

[25] Jeff Y-C Pan and Jay M. Tenenbaum. An intelligent agent framework for enterprise integration. *IEEE Transactions on Systems, Man and Cybernetics*, 21(6), December 1991. (Special Issue on Distributed AI).

[26] Jon Pastor, Don McKay, and Tim Finin. Viewconcepts: Knowledge-based access to databases. In *First International Conference on Information and Knowledge Management*, October 1992.

[27] R. Patil, R. Fikes, P. Patel-Schneider, D. McKay, T. Finin, T. Gruber, and R. Neches. The DARPA knowledge sharing effort: Progress report. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, November 1992. Available as `http://www.cs.umbc.edu/kqml/papers/kr92.ps`.

[28] R. Patil, R. Fikes, P. Patel-Schneider, D. McKay, T. Finin, T. Gruber, and R. Neches. The DARPA knowledge sharing effort: Progress report. In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proc. of the Third International Conference (KR'92)*, San Mateo, CA, November 1992. Morgan Kaufmann.

[29] Gio Wiederhold, Peter Wegner, and Stefano Ceri. Toward megaprogramming. *Communications of the ACM*, 33(11):89–99, November 1992.

[30] David Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann Publishers, Inc., San Mateo, CA., 1988.

Software Agents (1994) (Make Corrections) (153 citations)
Michael R. Genesereth, Steven P. Ketchpel
Communications of the ACM

View or download:
ai.univie.ac.at/%7Epaolo...agents.ps.gz
aragorn.wirtschaft...twareagents.ps.gz
wachau.ai.univie.ac.at/%...agents.ps.gz
Cached: PS.gz PS PDF DjVu Image Update Help

From: ai.univie.ac.at/~paolo/lva/vu... (more)
From: wachau.ai.univie.ac.at/~paolo/...
(Enter author homepages)

Rate this article: 1 2 3 4 5 (best)
View Comments (0)

Abstract: this paper, we discuss these questions and describe some emerging technologies that provide answers. In the final section, we mention some additional issues and summarize the key points of the paper. (For more information on agent-based software engineering, see [Genesereth 1989] and [Genesereth 1992]. See also [Shoham 1993] for a description of a variation of agent-based software engineering known as "agent-oriented programming".) 2. Agent Communication Language (Update)

Context of citations to this paper: More

...among events, actions, and goals. **Moreover, knowledge can be exchanged with other agents, or increased by some inferential activity [21].** Although mobility is not the most characterizing aspect of these entities [22] there is a tendency to blend this notion of intelligent...

...dialogue among the agents involved. **All these elements provide for effective knowledge exchange among agents in heterogeneous environments [2].**The abstract FIPA architecture [3] provides mechanisms that can be used to enact the communication process among heterogeneous agent...

Cited by: More
A Formal Architectural Model - For Logical Agent (Correct)
The Eva Teleteaching Project - The - Concept And The (Correct)
InfoSleuth: Agent-Based Semantic Integration of.. - Bayardo, Jr.. (1997) (Correct)

Similar documents (at the sentence level): More
42.8%: A Distributed and Anonymous Knowledge Sharing Approach to.. - Michael Genesereth (1994) (Correct)
35.0%: A Distributed And Anonymous Knowledge Sharing Approach To.. - Singh, Genesereth (1994) (Correct)
10.1%: Coordinating Distributed Objects With Declarative Interfaces - Narinder Singh (1995) (Correct)

Active bibliography (related documents): More All
0.4: Eye On The Prize - Nilsson (1995) (Correct)
0.2: Towards Megaprogramming: A Paradigm for Component-Based.. - Gio Wiederhold (1992) (Correct)
0.2: Software Agents and Intelligent Object Fusion - Nourani (1997) (Correct)

Similar documents based on text: More All
0.1: Anthropologically-Based Migration of Agents: a New Approach.. - Bordini, Campbell (Correct)
0.1: The Networked Information Economy: Applied And Theoretical.. - And The (Correct)
0.1: Towards Secure Mediation - Biskup, Flegel, Karabulut (1998) (Correct)

Related documents from co-citation: More All
26: KQML as an Agent Communication Language - Finin - 1994
22: Agents that reduce work and information overload (context) - Maes - 1994
17: Enabling technology for knowledge sharing (context) - Neches, Fikes et al. - 1991

BibTeX entry: (Update)

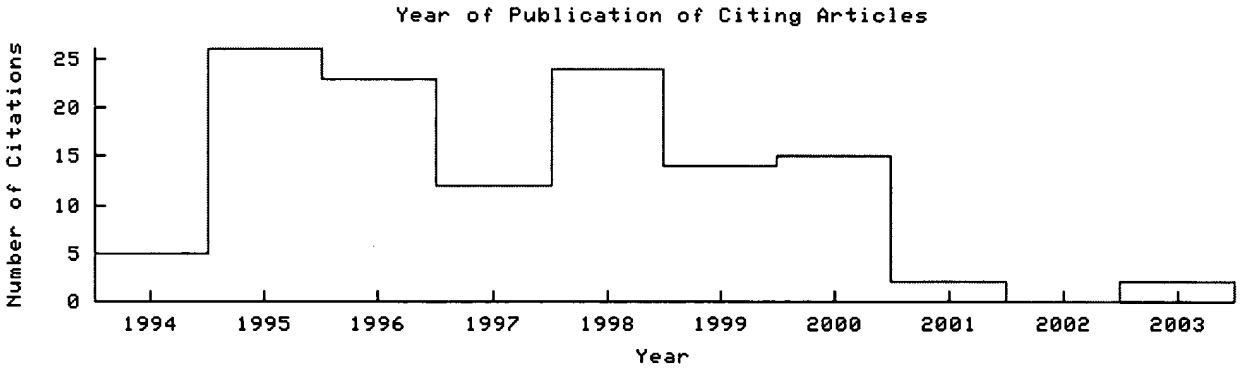
Genesereth, M. R., and Ketchpel, S. P., "Software Agents," Communication of the ACM, Vol. 37, No. 7 July 1994. http://citeseer.ist.psu.edu/genesereth94software.html More

```
@article{ genesereth97software,
```

```
author = "Michael R. Genesereth and Steven P. Ketchpel",
title = "Software Agents",
journal = "Communications of the {ACM}",
volume = "37",
number = "7",
year = "1997",
url = "citeseer.ist.psu.edu/genesereth94software.html" }
```

Citations (may not include all citations):

396 Agent-Oriented Programming (context) - Shoham - 1993
187 Enabling Technology for Knowledge Sharing (context) - Neches, Fikes et al. - 1991
185 Negotiation as a Metaphor for Distributed Problem Solving (context) - Davis, Smith - 1983
159 A Market-Oriented Programming Environment and its Applicatio.. - Wellman - 1993
92 Ontolingua: A Mechanism to Support Portable Ontologies - Gruber - 1991
68 Deals Among Rational Agents (context) - Rosenschein, Genesereth - 1985
44 A Decision- Theoretic Approach to Coordinating Multiagent In.. (context) - Gmytrasiewicz, Durfee et al. - 1991
38 An Overview of KQML: A Knowledge Query and Manipulation Lang.. (context) - Finin, Wiederhold - 1991
35 Agents Contracting Tasks in Non-Collaborative Environments (context) - Kraus - 1993
31 The Clarke Tax as a consensus mechanism among automated agen.. (context) - Ephrati, Rosenschein - 1991
22 Understanding the Role of Negotiation in Distributed Search .. - Lander, Lesser - 1993
14 An Agent-Based Approach to Software Interoperability (context) - Genesereth - 1992
10 Knowledge Interchange Format Version 3 Reference Manual - Genesereth, Fikes - 1992
6 The Architecture of Future Information Systems (context) - Wiederhold - 1989
4 PACT: An Experiment in Integrated Engineering Systems (context) - Cutkosky - 1993
4 A Proposal for Research on Informable Agents (context) - Genesereth - 1989
3 Mechanisms for Automated Negotiation among Autonomous Agents (context) - Zlotkin - 1994



The graph only includes citing articles where the year of publication is known.

Documents on the same site (http://www.ai.univie.ac.at/~paolo/lva/vu-sa/): More
INTELLIGENT AGENTS ON THE INTERNET: Fact, Fiction, and Forecast - Etzioni, Weld (1995) (Correct)
COLLAGEN: When Agents Collaborate with People - Rich, Sidner (1996) (Correct)
Letizia: An Agent That Assists Web Browsing - Lieberman (1995) (Correct)

Software Agents

Michael R. Genesereth
Logic Group
Computer Science Department
Stanford University

Steven P. Ketchpel
Computer Science Department
Stanford University

1. Introduction

The software world is one of great richness and diversity. Many thousands of software products are available to users today, providing a wide variety of information and services in a wide variety of domains. While most of these programs provide their users with significant value when used in isolation, there is increasing demand for programs that can *interoperate* – to exchange information and services with other programs and thereby solve problems that cannot be solved alone.

Part of what makes interoperation difficult is heterogeneity. Programs are written by different people, at different times, in different languages; and, as a result, they often provide different interfaces. The difficulties created by heterogeneity are exacerbated by dynamics in the software environment. Programs are frequently rewritten; new programs are added; old programs removed.

Agent-based software engineering was invented to facilitate the creation of software able to interoperate in such settings. In this approach to software development, application programs are written as *software agents*, i.e. software "components" that communicate with their peers by exchanging messages in an expressive *agent communication language.*

Agents can be as simple as subroutines; but typically they are larger entities with some sort of persistent control (e.g. distinct control threads within a single address space, distinct processes on a single machine, or separate processes on different machines).

The salient feature of the language used by agents is its expressiveness. It allows for the exchange of data and logical information, individual commands and scripts (i.e. programs). Using this language, agents can communicate complex information and goals, directly or indirectly "programming" each other in useful ways.

Agent-based software engineering is often compared to object-oriented programming. Like an "object", an agent provides a message-based interface independent of its internal data structures and algorithms. The primary difference between the two approaches lies in the language of the interface. In general object-oriented programming, the meaning of a message can vary from one object to another. In agent-based software engineering, agents use a common language with an agent-independent semantics.

The concept of agent-based software engineering raises a number of important questions.

(1) What is an appropriate agent communication language?

(2) How do we build agents capable of communicating in this language?

(3) What communication "architectures" are conducive to cooperation?

In the next three sections of this paper, we discuss these questions and describe some emerging technologies that provide answers. In the final section, we mention some additional issues and summarize the key points of the paper. (For more information on agent-based software engineering, see [Genesereth 1989] and [Genesereth 1992]. See also [Shoham 1993] for a description of a variation of agent-based software engineering known as "agent-oriented programming".)

2. Agent Communication Language

Communication language standards facilitate the creation of interoperable software by decoupling implementation from interface. So long as programs abide by the details of the standards, it does not matter how they are implemented. Today, standards exist for a wide variety of domains. For example, electronic mail programs from different vendors manage to interoperate through the use of mail standards like SMTP. Disparate graphics programs interoperate using standard formats like GIF and JPEG. Text formatting programs and printers interoperate using languages like PostScript.

Unfortunately, problems arise when it becomes necessary for programs that use one language to interoperate with programs that use a different language. To begin with, there can be *inconsistencies* in the use of syntax or vocabulary. One program may use a word or expression to mean one thing while another program uses the same word or expression to mean something entirely different. At the same time, there can be *incompatibilities*. Different programs may use different words or expressions to say the same thing.

Agent-based software engineering attacks these problems by mandating a universal communication language, one in which inconsistencies and arbitrary notational variations are eliminated. There are two popular approaches to the design of such a language – the procedural approach and the declarative approach.

The procedural approach is based on the idea that communication can be best modelled as the exchange of procedural directives. Scripting languages (such as TCL, Apple Events, and Telescript) are based on this approach. They are both simple and powerful. They allow programs to transmit not only individual commands but entire programs, thus implementing delayed or persistent goals of various sorts. They are also (usually) directly and efficiently executable.

Unfortunately, there are disadvantages to purely procedural languages. For one, devising procedures sometimes requires information about the recipent that may not be available to the sender. Secondly, procedures are unidirectional. Much information that agents must share should be usable in both directions – to compute quantity a from quantity b at one time and to compute quantity b from quantity a at another. Most significantly, scripts are difficult to merge. This is no problem so long as all communication is one-on-one. However, things become more difficult when an agent receives multiple scripts from multiple agents that must be run simultaneously and may interfere with each other. Merging procedural information is much more difficult than merging declarative specifications or mixed mode information (like condition-action rules).

In contrast with this procedural approach, the declarative approach to language design is based on the idea that communication can be best modelled as the exchange of declarative statements (definitions, assumptions, and the like). To be maximally useful, a

declarative language must be sufficiently expressive to communicate information of widely varying sorts (including procedures). At the same time, the language must be reasonably compact; it must ensure that communication is possible without excessive growth over specialized languages. As an exploration of this approach to communication, researchers in the ARPA Knowledge Sharing Effort [Neches] have defined the components of an agent communication language (called ACL) that satisfies these needs.

ACL can best be thought of as consisting of three parts – its vocabulary, an "inner language" called KIF (short for Knowledge Interchange Format), and an "outer" language called KQML (short for Knowledge Query and Manipulation Language). An ACL message is a KQML expression in which the "arguments" are terms or sentences in KIF formed from words in the ACL vocabulary.

The vocabulary of ACL is listed in a large and open-ended dictionary of words appropriate to common application areas [Gruber]. Each word in the dictionary has an English description for use by humans in understanding the meaning of the word; and each word has formal annotations (written in KIF) for use by programs. The dictionary is open-ended to allow for the addition of new words within existing areas and in new application areas.

Note that the existence of such a dictionary does not imply that there is only one way of describing an application area. Indeed, the dictionary can contain multiple *ontologies* for any given area. For example, it contains vocabulary for describing three dimensional geometry in terms of polar coordinates, rectangular coordinates, cylindrical coordinates, etc. A program can use whichever ontology is most convenient. The formal definitions of the words associated with any one of these ontologies can then be used by system programs in translating messages using one ontology into messages using other ontologies.

KIF is a prefix version of first order predicate calculus, with various extensions to enhance its expressiveness. It provides for the encoding of simple data, constraints, negations, disjunctions, rules, quantified expressions, metalevel information, and so forth. See figure 1 for a brief summary of KIF.

While it is possible to design an entire communication framework in which all messages take the form of KIF sentences, this would be efficient. Because of the contextual independence of KIF's semantics, each message would have to include any implicit information about the sender, the receiver, the time of the message, message history, and so forth. The efficiency of communication can be enhanced by providing a linguistic layer in which context is taken into account. This is the function of KQML. See figure 2 for a brief summary.

ACL has been used in several large-scale demonstrations of software interoperation, and the results are promising. Full specifications are available, and parts of the language are making their way through various standards organizations. Several start-up companies are proposing to offer commercial products for processing ACL; and a number of established computer system vendors are looking at ACL as a possible language for communication among heterogeneous systems.

As of this writing, it is not clear which of these two approaches will succeed. The declarative approach seems inevitable in the long run. However, scripting languages are likely to be popular in the short run because of their familiarity; and so the ultimate agent communication language may end up looking more like a scripting language than ACL.

Figure 1 – Knowledge Interchange Format

KIF [Genesereth, Fikes, et al.] is a prefix version of the language of first order predicate calculus with various extensions to enhance its expressiveness.

First and foremost, KIF provides for the expression of simple data. For example, the sentences shown below encode 3 tuples in a personnel database. The first argument in each is the social security number of an individual, the second argument is the department within which the individual works, and the third argument is the individual's salary.

```
(salary 015-46-3946 widgets 72000)
(salary 026-40-9152 grommets 36000)
(salary 415-32-4707 fidgets 42000)
```

More complicated pieces of information can be expressed through the use of complex terms. For example, the following sentences states that one chip is larger than another.

```
(> (* (width chip1) (length chip1)) (* (width chip2) (length chip2)))
```

KIF includes a variety of logical operators to assist in the encoding of logical information (such as negations, disjunctions, rules, quantified formulas, and so forth). The expression shown below is an example of a complex sentence in KIF. It asserts that the number obtained by raising any real-number `?x` to an even power `?n` is positive.

```
(=> (and (real-number ?x) (even-number ?n)) (> (expt ?x ?n) 0))
```

One of the distinctive features of KIF is its ability to encode knowledge about knowledge, using the ' and , operators and related vocabulary. For example, the following sentence asserts that agent Joe is interested in receiving triples in the salary relation. The use of commas signals that the variables should not be taken literally. Without the commas, this sentence would say that agent 1 is interested in the sentence `(salary ?x ?y ?z)` instead of its instances.

```
(interested joe '(salary ,?x ,?y ,?z))
```

KIF can also be used to describe procedures, i.e. to write programs or scripts for agents to follow. Given the prefix syntax of KIF, such programs resemble Lisp or Scheme. The following is an example of a three step procedure written in KIF. The first step ensures that there is a fresh line on the standard output stream; the second step is to print `Hello!` to the standard output stream; the final step is to add a carriage return to get to a new fresh line.

```
(progn (fresh-line t) (print "Hello!") (fresh-line t))
```

The semantics of the KIF core (KIF without rules and definitions) is similar to that of first order logic. There is an extension to handle nonstandard operators (like ' and ,), and there is a restriction to models that satisfy various axiom schemata (to give meaning to the basic vocabulary in the format). Despite these extensions and restrictions, the core language retains the fundamental characteristics of first order logic, including compactness and the semidecidability of logical entailment.

Figure 2 – Knowledge Query and Manipulation Language

As used in ACL, KQML *messages* are similar to KIF expressions. Each message is a list of components enclosed in matching parentheses. The first word in the list indicates the type of communication. The subsequent entries are KIF expressions appropriate to that communication, in effect the "arguments".

Intuitively, each message in KQML is one piece of a dialog between between the sender and the receiver, and KQML provides support for a wide variety of such dialog types.

The expression shown below is the simplest possible KQML dialog. In this case, there is just one message – a simple notification. The sender is conveying the enclosed sentence to the receiver. In general, there is no expectation on the sender's part about what use the receiver will make of this information.

```
A to B: (tell (> 3 2))
```

The following dialog is a little more interesting. In this case, the first message is a request for the receiver to execute the operation of printing a string to its standard i/o stream. The second message tells the sender that the request has been satisfied.

```
A to B: (perform (print "Hello!" t))
B to A: (reply done)
```

In the dialog shown below, the sender is asking the receiver a question in an `ask-if` message. The receiver then sends the answer to the original sender in a `reply` message.

```
A to B: (ask-if (> (size chip1) (size chip2)))
B to A: (reply true)
```

In the following case, the sender asks the receiver to send it a notification whenever it receives information about the position of an object. The receiver sends it three such sentences, after which the original sender cancels the service.

```
A to B: (subscribe (position ?x ?r ?c))
B to A: (tell (position chip1 8 10))
B to A: (tell (position chip2 8 46))
B to A: (tell (position chip3 8 64))
A to B: (unsubscribe (position ?x ?r ?c))
```

In addition to simple notifications, commands, questions, and subscriptions, as illustrated here, KQML also contains support for delayed and conditional operations, requests for bids, offers, promises, and so forth.

(For those who have seen a little of KQML and wonder where the packages went, it is worth noting that, in addition to the communication layer described here, KQML includes yet another linguistic layer to support the transmission of messages among agents operating in different processes. This layer characterizes the additional information that must be conveyed in communication protocols between distributed systems, such as email and TCP connections. The details of this "package" layer are irrelevant to the discussion in this paper; see the KQML document for more information.)

3. Agents

The criterion for agenthood is a behavioral one. An entity is a software agent if and only if it communicates correctly in an agent communication language like ACL. This means that the entity must be able to read and write ACL messages, and it means that the entity must abide by the behavioral constraints implicit in the meanings of those messages.

The specific constraints associated with a message derive from the content of that message and general principles of agent behavior. For example, there is veracity (an agent must tell the truth), autonomy (an agent may not constrain another agent to perform a service unless the other agent has advertised its willingness to accept such a request), commitment (if an agent advertises a willingness to perform a service, then it is obliged to perform that service when asked to do so), and so forth.

From a theoretical perspective, it is interesting to note that all of these principles can be derived from the single principle of veracity. In other words, if all agents are constrained to tell the truth, then autonomy, commitment, etc. all follow. To many people, the principle of veracity sounds too strong; but it is not difficult to achieve. An agent can always state its own inputs, outputs, and definitions with confidence; and it can nest conjectures inside of statements about its "beliefs". Unfortunately, a full account of this issue is beyond the scope of this paper; and, interesting as it may be theoretically, it has only indirect practical value.

For our purposes here, it is sufficient to say that the use of ACL brings with it behavioral constraints. However, this leaves opens a wide range of possibilities. At one extreme, we can imagine "perfect" agents that retain all of the information they receive and act in accordance with the logical consequences of this information. At the other extreme, we can imagine simple agents, like calculators, that answer arithmetic problems and ignore everything else. More powerful agents utilize a larger portion of ACL; less powerful agents use a smaller subset. All are agents, so long as they use the language correctly.

Given a clear statement of the language and the behavioral principles that agents must satisfy, it is straightforward to write programs that behave correctly. But what about all of the programs that have already been written, our so-called "legacy" software? Are there any standard techniques for converting such programs into software agents? In work thus far, a number of different approaches have been taken. See figure 3.

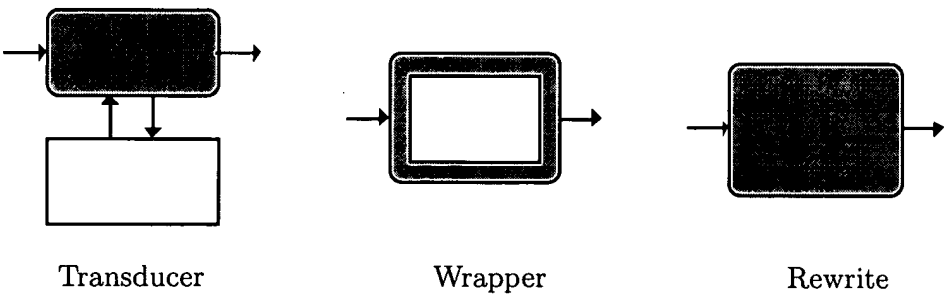


Figure 3 - Three approaches to agentification

One approach (the leftmost diagram in figure 3) is to implement a *transducer* that mediates between an existing program and other agents. The transducer accepts messages from other agents, translates them into the program's native communication protocol, and passes those messages to the program. It accepts the program's responses, translates into ACL, and sends the resulting messages on to other agents.

This approach has the advantage that it requires no knowledge of the program other than its communication behavior. It is, therefore, especially useful for situations in which the code for the program is unavailable or too delicate to modify.

This approach also works for other types of resources, such as files and people. It is a simple matter to write a program to read or modify an existing file with a specialized format and thereby provide access to that file via ACL. Similarly, it is possible to provide a graphical user interface for a person that allows that person to interact with the system in a specialized graphical language, which is then converted into ACL, and vice versa.

A second approach to dealing with legacy software (the middle diagram in figure 3) is to implement a *wrapper*, i.e. inject code into a program to allow it to communicate in ACL. The wrapper can directly examine the data structures of the program and can modify those data structures. Furthermore, it may be possible to inject calls out of the program so that it can take advantage of externally available information and services.

This approach has the advantage of greater efficiency than the transduction approach, since there is less serial communication. It also works for cases where there is no interprocess communication ability in the original program. However, it requires that the code for the program be available.

The third and most drastic approach to dealing with legacy software (the rightmost diagram in figure 3) is to rewrite the original program. The advantage of this approach is that it may be possible to enhance its efficiency or capability beyond what would be possible in either the transduction or wrapping approaches.

The best examples of this approach come from the engineering domain. Many automated design programs work to completion before communicating with other programs. For example, the output of a logic synthesis program is passed as input to a printed circuit board layout and routing program; its output is passed to an assembly planning program; and so forth. Recent work in concurrent engineering suggests that there is much advantage to be gained by writing programs that communicate partial results in the course of their activity and that accept partial results and feedback from other programs. By communicating a partial result and getting early feedback, a program can save work on what may turn out to be an unworkable alternative.

4. Architecture of Multi-Agent Systems

Once we have a language and the ability to build agents, there remains the question of how these agents should be organized to enhance collaboration. Two very different approaches have been explored: direct communication (in which agents handle their own coordination) and assisted coordination (in which agents rely on special system programs to achieve coordination).

The advantage of direct communication is that it does not rely on the existence, capabilities, or biases of any other programs. Two popular architectures for direct com-

munication are the contract-net approach and specification sharing.

In the contract net approach to interoperation [Davis and Smith 1983], agents in need of services distribute *requests for proposals* to other agents. The recipients of these messages evaluate those requests and submit *bids* to the originating agents. The originators use these bids to decide which agents to task and then award *contracts* to those agents.

In the specification sharing approach to interoperation, agents supply other agents with information about their *capabilities* and *needs*; and these agents can then use this information to coordinate their activities. The specification sharing approach is often more efficient than the contract net approach because it decreases the amount of communication that must take place.

One disadvantage of direct communication is cost. So long as the number of agents is small, this is not a problem. But, in a setting like the Internet, with millions of programs, the cost of broadcasting bids or specifications and the consequential processing of those messages is prohibitive. In this case, the only alternative is to organize the agents in some way that avoids such broadcasts.

Another disadvantage is implementational complexity. In the direct communication schemes, each agent is responsible for negotiating with other agents and must contain all of the code necessary to support this negotiation. If only these capabilities could be provided by the system, this would lessen the complexity of application programs.

A popular alternative to direct communication that eliminates both of these disadvantages is to organize agents into what is often called a *federated system*. Figure 4 illustrates the structure of such a system in the simple case in which there are just three machines, one with three agents and two with two agents apiece. As suggested by the diagram, agents do not communicate directly with each other. Instead, they communicate only with system programs called *facilitators*, and facilitators communicate with each other. (The concept of a facilitator [Genesereth 1992] derives from and generalizes the concept of a *mediator* [Wiederhold].)

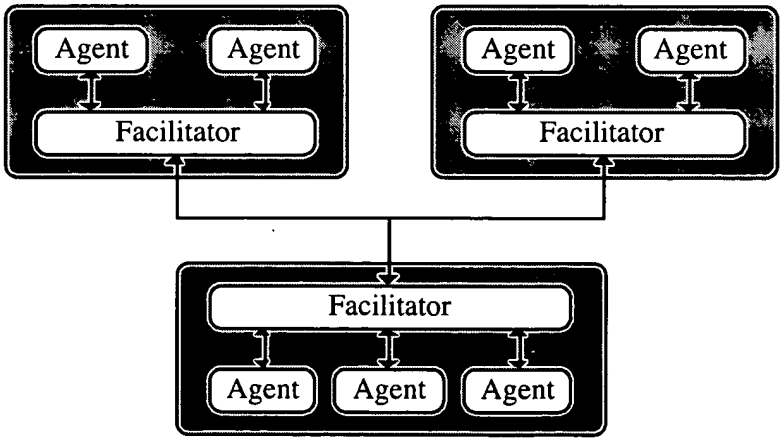


Figure 4 - Federated system

In a federated system, agents use ACL (in practice, a restricted subset of ACL) to document their needs and abilities for their local facilitators. In addition to this metalevel

information, they also send application-level information and requests to their facilitators and accept application-level information and requests in return. Facilitators use the documentation provided by these agents to transform these application-level messages and route them to the appropriate places. In effect, the agents form a "federation" in which they surrender their autonomy to their facilitators and the facilitators take the responsibility for fulfilling their needs.

The concepts of system services in support of software interoperation is not new here. For example, directory assistance programs facilitate software interoperation by providing a way for programs to discover which programs can handle which requests and which programs are interested in which pieces of information. Distributed object managers (like CORBA, OLE, DSOM) provide location transparency for object-oriented systems, routing messages to objects without requiring senders to know the locations of those objects. Automatic brokers (like the Publish and Subscribe capabilities on the Macintosh, DDE, BMS, Tooltalk, etc.) combine these capabilities – they not only compute the appropriate programs to receive messages but forward those messages, handle any problems that arise, and, where appropriate, return the answers to the original senders.

The primary difference between these approaches to software interoperation and agent-based software engineering lies in the sophistication of the processing done by facilitators. Using ACL, agents can express their needs and capabilities more accurately than in pattern-based metalanguages; and facilitators can use this added information to be more discriminating in routing messages. In order to deal with notational incompatibilities, facilitators can translate messages from one vocabulary to another using definitions supplied by agents or retrieved from the ACL dictionary. In so doing, they can decompose messages into sub-messages and send them to different agents. When necessary, they can combine multiple messages. In some cases, this assistance can be rendered interpretively (with messages going through the facilitators); in other cases, it can be done in one-shot fashion (with the facilitators setting up specialized links between individual agents and then stepping out of the picture).

In order to provide these capabilities, current implementations of facilitators take advantage of automated reasoning technology developed in the Artificial Intelligence and Database communities. Powerful search control techniques are used to enhance normal message-processing performance; and automatic generation of message routing programs and pairwise translators is used for cases requiring greater efficiency.

Even with these enhancements, these implementations consume more time in the worst case than simpler processing techniques (like the pattern matching method used in BMS). This is sometimes acceptable, especially when the alternative is no interoperation at all. However, in time critical applications (such as machine control), the extra cost can be prohibitive.

5. Summary

The agent-based approach to software interoperation described here has been developed into a practical technology and has been put to use in a variety of applications necessitating interoperation (e.g. concurrent engineering [Cutkosky], database integration, and so forth) and is being used at multiple institutions in the construction of software for

the national information infrastructure.

In order to concentrate on the central issues in agent-based software engineering, we have ignored many key problems in our presentation, such as synchronization, security, payment for services, crash recovery, inconsistencies in program specifications, and so forth. Although partial solutions to these problems exist, further work is needed.

In our treatment so far, we have assumed that there is sufficient common interest among the agents that they will frequently volunteer to help each other and receive no direct reward for their labor. As the Internet becomes increasingly commercialized, we envision a world where agents act on behalf of their creators to make a profit. Agents will seek payment for services provided and may negotiate with each other to maximize their expected utility, which might be measured in a form of electronic currency.

These problems mark the intersection of economics and distributed artificial intelligence (DAI). A number of researchers in DAI are using tools developed in economics and game theory to evaluate multi-agent interactions [Zlotkin 1994], [Rosenschein and Genesereth 1985], [Gmytrasiewicz, Durfee and Wehe 1991]. Depending on the prevailing conditions of the situation, any one of a number of protocols might be applicable. In the simplest case, the agent requesting a service offers a specific reward for the completion of a task. The agent that performs the task receives the payment. In more complex scenarios, a task may be completed by a set of agents, who need to negotiate how to divide the reward. Dividing the total amount equally might not be fair if the agents made different contributions. If there are many agents (or sets of agents) that may complete the task, the requestor might try to minimize its cost by seeking multiple bids or holding an auction. There are a number of alternatives (e.g. English Ascending Auction, Dutch Descending Auction, Sealed-Bid, Vickery's Second Price) that have different properties and may be applicable or preferred in different situations. The WALRAS system [Wellman 1993] is an example of market mechanics being used to coordinate agents.

A further goal of DAI research is to obviate the need for the truth-telling assumption. If the selected protocols are truth dominant, agents tell the truth out of self-interest, rather than by fiat. This makes the system as a whole more resistant to a scheming agent that might try to exploit other agents by lying. The next step in this research thread is to create protocols that are resistant to the efforts of groups of agents that attempt to manipulate the system for their own benefit.

In this paper, we have taken a brief look at how agent technology can be used to promote software interoperation. Our long-range vision is one in which any system (software or hardware) can interoperate with any other system, without the intervention of human users or their programmers. Although many problems remain to be solved, we believe that the introduction of agent technology will be an important step toward achieving this vision.

References

1. Cutkosky, M. et al. PACT: An Experiment in Integrated Engineering Systems, *Computer 26*, 1(1993), 28-37.

2. Davis, R., and Smith, R. G. Negotiation as a Metaphor for Distributed Problem Solving, in *Artificial Intelligence 20*, 1(1983), 63-109.

3. Ephrati, E. and Rosenschein, J. S. The Clarke Tax as a consensus mechanism among automated agents". In *Proceedings of the Ninth National Conference on Artificial Intelligence* (Anaheim, California 1991). AAAI Press, Menlo Park, CA, pp. 173-178.

4. Finin, T., and Wiederhold, G. An Overview of KQML: A Knowledge Query and Manipulation Language, available through the Stanford University Computer Science Department, 1991.

5. Genesereth, M. R., Fikes, R. E. et al. Knowledge Interchange Format Version 3 Reference Manual, Logic-92-1, Stanford University Logic Group, 1992.

6. Genesereth, M. R. A Proposal for Research on Informable Agents, Logic-89-9, Stanford University Logic Group, June 1989.

7. Genesereth, M. R. An Agent-Based Approach to Software Interoperability, In *Proceedings of the DARPA Software Technology Conference*, 1992.

8. Gmytrasiewicz, P. J., Durfee, E. H. and Wehe, D. K. A Decision- Theoretic Approach to Coordinating Multiagent Interactions. In *Proceedings of the Twelfth International Joint Conference On Artificial Intelligence* (Sydney, Australia 1991). International Joint Conferences on Artificial Intelligence, Inc. pp. 62-68.

9. Gruber, T. Ontolingua: A Mechanism to Support Portable Ontologies, KSL-91-66, Stanford Knowledge Systems Laboratory, 1991.

10. Kraus, S. Agents Contracting Tasks in Non-Collaborative Environments. In *Proceedings of the Eleventh National Conference on Artificial Intelligence* (Washington, DC 1993). AAAI Press, Menlo Park, CA. pp. 243-248.

11. Lander, S. E. and Lesser, V. R. Understanding the Role of Negotiation in Distributed Search Among Heterogeneous Agents. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (Chambery, France 1993). International Joint Conferences on Artificial Intelligence, Inc. pp. 438-444.

12. Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., and Swartout, W. Enabling Technology for Knowledge Sharing, *AI Magazine 12*, 3(1991), 36-56.

13. Rosenschein, J. S. and Genesereth, M.R. Deals Among Rational Agents. In *Proceed-*

ings of the Ninth International Joint Conference on Artificial Intelligence (Los Angelos, California 1985). AAAI Press, Menlo Park, CA, pp. 91-99.

14. Shoham, Y. Agent-Oriented Programming. *Artificial Intelligence 60.* 1(1993), 51-92.

15. Wellman, M. P. A Market-Oriented Programming Environment and its Application to Distributed Multicommodity Flow Problems, *Journal of Artificial Intelligence Research 1* (1993), 1-23.

16. Wiederhold, G. The Architecture of Future Information Systems, Stanford University Computer Science Department, 1989.

17. Zlotkin, G. Mechanisms for Automated Negotiation among Autonomous Agents. Ph.D. Dissertation. Hebrew University. February 1994.

Software Agents (1994) (Make Corrections) (153 citations)
Michael R. Genesereth, Steven P. Ketchpel
Communications of the ACM

(Enter summary)

Abstract: this paper, we discuss these questions and describe some emerging technologies that provide answers. In the final section, we mention some additional issues and summarize the key points of the paper. (For more information on agent-based software engineering, see [Genesereth 1989] and [Genesereth 1992]. See also [Shoham 1993] for a description of a variation of agent-based software engineering known as "agent-oriented programming".) 2. Agent Communication Language (Update)

Context of citations to this paper: More

...among events, actions, and goals. **Moreover, knowledge can be exchanged with other agents, or increased by some inferential activity [21].** Although mobility is not the most characterizing aspect of these entities [22] there is a tendency to blend this notion of intelligent...

...dialogue among the agents involved. **All these elements provide for effective knowledge exchange among agents in heterogeneous environments [2].**The abstract FIPA architecture [3] provides mechanisms that can be used to enact the communication process among heterogeneous agent...

Cited by: More
A Formal Architectural Model - For Logical Agent (Correct)
The Eva Teleteaching Project - The - Concept And The (Correct)
InfoSleuth: Agent-Based Semantic Integration of.. - Bayardo, Jr.. (1997) (Correct)

Similar documents (at the sentence level): More
42.8%: A Distributed and Anonymous Knowledge Sharing Approach to.. - Michael Genesereth (1994) (Correct)
35.0%: A Distributed And Anonymous Knowledge Sharing Approach To.. - Singh, Genesereth (1994) (Correct)
10.1%: Coordinating Distributed Objects With Declarative Interfaces - Narinder Singh (1995) (Correct)

Active bibliography (related documents): More All
0.4: Eye On The Prize - Nilsson (1995) (Correct)
0.2: Towards Megaprogramming: A Paradigm for Component-Based.. - Gio Wiederhold (1992) (Correct)
0.2: Software Agents and Intelligent Object Fusion - Nourani (1997) (Correct)

Similar documents based on text: More All
0.1: Anthropologically-Based Migration of Agents: a New Approach.. - Bordini, Campbell (Correct)
0.1: The Networked Information Economy: Applied And Theoretical.. - And The (Correct)
0.1: Towards Secure Mediation - Biskup, Flegel, Karabulut (1998) (Correct)

Related documents from co-citation: More All
26: KQML as an Agent Communication Language - Finin - 1994
22: Agents that reduce work and information overload (context) - Maes - 1994
17: Enabling technology for knowledge sharing (context) - Neches, Fikes et al. - 1991

BibTeX entry: (Update)

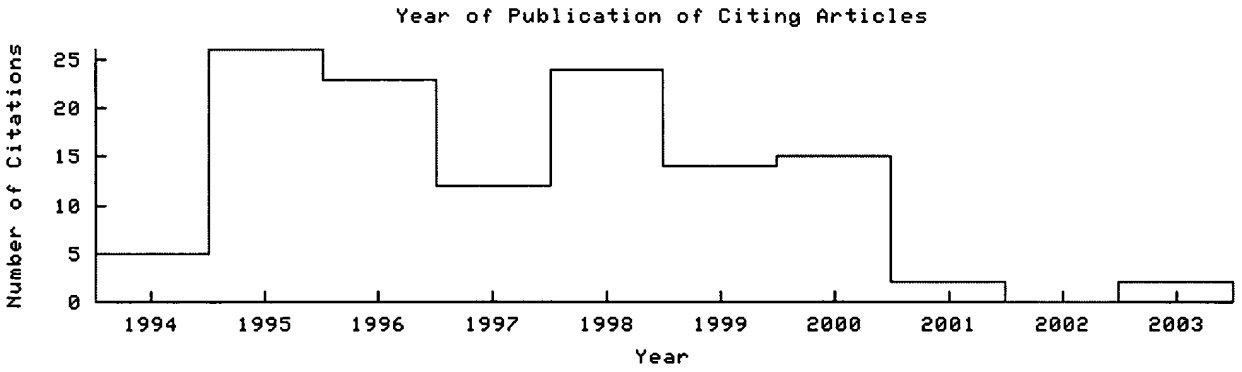
Genesereth, M. R., and Ketchpel, S. P., "Software Agents," Communication of the ACM, Vol. 37, No. 7 July 1994. http://citeseer.ist.psu.edu/genesereth94software.html More

```
@article{ genesereth97software,
```

```
author = "Michael R. Genesereth and Steven P. Ketchpel",
title = "Software Agents",
journal = "Communications of the {ACM}",
volume = "37",
number = "7",
year = "1997",
url = "citeseer.ist.psu.edu/genesereth94software.html" }
```

Citations (may not include all citations):

396 Agent-Oriented Programming (context) - Shoham - 1993
187 Enabling Technology for Knowledge Sharing (context) - Neches, Fikes et al. - 1991
185 Negotiation as a Metaphor for Distributed Problem Solving (context) - Davis, Smith - 1983
159 A Market-Oriented Programming Environment and its Applicatio.. - Wellman - 1993
92 Ontolingua: A Mechanism to Support Portable Ontologies - Gruber - 1991
68 Deals Among Rational Agents (context) - Rosenschein, Genesereth - 1985
44 A Decision- Theoretic Approach to Coordinating Multiagent In.. (context) - Gmytrasiewicz, Durfee et al. - 1991
38 An Overview of KQML: A Knowledge Query and Manipulation Lang.. (context) - Finin, Wiederhold - 1991
35 Agents Contracting Tasks in Non-Collaborative Environments (context) - Kraus - 1993
31 The Clarke Tax as a consensus mechanism among automated agen.. (context) - Ephrati, Rosenschein - 1991
22 Understanding the Role of Negotiation in Distributed Search .. - Lander, Lesser - 1993
14 An Agent-Based Approach to Software Interoperability (context) - Genesereth - 1992
10 Knowledge Interchange Format Version 3 Reference Manual - Genesereth, Fikes - 1992
6 The Architecture of Future Information Systems (context) - Wiederhold - 1989
4 PACT: An Experiment in Integrated Engineering Systems (context) - Cutkosky - 1993
4 A Proposal for Research on Informable Agents (context) - Genesereth - 1989
3 Mechanisms for Automated Negotiation among Autonomous Agents (context) - Zlotkin - 1994



The graph only includes citing articles where the year of publication is known.

Documents on the same site (http://www.ai.univie.ac.at/~paolo/lva/vu-sa/): More
INTELLIGENT AGENTS ON THE INTERNET: Fact, Fiction, and Forecast - Etzioni, Weld (1995) (Correct)
COLLAGEN: When Agents Collaborate with People - Rich, Sidner (1996) (Correct)
Letizia: An Agent That Assists Web Browsing - Lieberman (1995) (Correct)

CiteSeer.IST

Scientific Literature Digital Library

Agent Communication Langauge

Search Documents | Search Citations

Documents indexed by CiteSeer.IST

Citations made by indexed documents

Announcements

Your search matched **5** of **1024576** documents.
A maximum of **500** results are displayed, **15** to a page, sorted by **Relevance Descending** order.

Refine This Search:
You may refine your search by editing the current search expression or enter new one in the text box.

multiagent <and> kqml Search

☐ Check to search within this result set

Results Key:
JNL = Journal or Magazine **CNF** = Conference **STD** = Standard

---

1 **Toward an open virtual market place for mobile agents**
Esmahi, L.; Dini, P.; Bernard, J.C.;
Enabling Technologies: Infrastructure for Collaborative Enterprises, 1999. (W ICE '99) Proceedings. IEEE 8th International Workshops on , 16-18 June 199
Pages:279 - 286

[Abstract] [PDF Full-Text (100 KB)] **IEEE CNF**

---

2 **A product retrieval system for electronic commerce based on KQML**
Jeong-Il Song; Han-Hyuk Chung; Eun-Seok Lee;
Parallel Processing, 1999. Proceedings. 1999 International Workshops on , 21 Sept. 1999
Pages:387 - 391

[Abstract] [PDF Full-Text (124 KB)] **IEEE CNF**

---

3 **Coordinating multiple agents in the supply chain**
Barbuceanu, M.; Fox, M.S.;
Enabling Technologies: Infrastructure for Collaborative Enterprises, 1996. Proceedings of the 5th Workshop on , 19-21 June 1996
Pages:134 - 141

[Abstract] [PDF Full-Text (1164 KB)] **IEEE CNF**

---

4 **Collaborative prototyping in distributed virtual reality using an age communication language**
Nedelec, A.; Reignier, P.; Rodin, V.;
Systems, Man, and Cybernetics, 2000 IEEE International Conference on , Vol 2 , 8-11 Oct. 2000
Pages:1007 - 1012 vol.2

[Abstract] [PDF Full-Text (508 KB)] IEEE CNF

5 **Developing coherent multiagent systems using JAFMAS**
Chauhan, D.; Baker, A.D.;
Multi Agent Systems, 1998. Proceedings. International Conference on , 3-7 Jı
1998
Pages:407 - 408

[Abstract] [PDF Full-Text (16 KB)] IEEE CNF

Agent Communication Language

By Haw Siang HON

ISE

At the wake of the Internet boom witnessed in the nineties, we are now witnessing the birth of a new branch of software phenomenon, that of agent-based systems.

The argument over the exact definition of an agent still rages on among the theorists. For the purpose of this article we simplify an agent by considering it as an entity that can decide for itself, and is autonomous in that it is a software program that does not require constant human control or supervision to carry out its set task. This, an autonomous agent, is what we will be referring to when we use the term, agent.

Agents are designed such that it executes its given task and fulfils its individual goal. However, as time passed, more complex tasks and problem evolved for agents-based system to solve. This led to the need for many agents to group together into a community of agents, or society, where they help each other by carrying out tasks for each other. As well as this, agents have also been used in intelligent systems that can negotiate and reason with fellow autonomous agent to reach agreements or to persuade one another to pursue a course of action, usually through a process of negotiation.

Either way, this has brought about the need of agent interactions and the issue of interoperability has become one of great significance. That is to allow for a community of agents to develop, reason and/or co-operate - to allow agents to exchange information and services with one another, as well as negotiating and reasoning against one another.

Many agents together in a community will form a society. Just like a real life society with humans, the need for a common medium for communication is essential for the agents to reason or co-operate with each other. The rise in popularity of agent based systems and greater demand for interoperability of agents have led to the need for a language that can be used not just in a proprietary domain, but inter domain, i.e., between different vendors over an network or internet. This will be the focus of this article, the wonderful world of **Agent Communication Language** or ACLs.

Communication techniques and protocol

Agent communication protocols or languages provides agents with a means of exchanging information and knowledge, which is really the essence of all forms of interaction in multi-agent systems. Such a protocol or language can be divided into three major components or layers, an "inner" and an "outer" language and its vocabulary.

The inner level entails the information content level, that is a logical language used to describe attitudes about their information or knowledge. This layer allows for knowledge sharing and it is the syntactic layer of knowledge representation. One of the main aims of such a language is to provide a common interchange format so that agent-dependent languages can easily be translated to and from this logical language. Using this language, an agent specifies its content.

The "vocabulary" or terminology is known as the ontology. This layer ensures that a term and indeed, even an object or entity, will have a uniform meaning amongst all agents involved in interaction even if different names are used for them. In short, ontology semantically unifies agent communication.

An agent uses the inner language to advertise its capability or its need in a co-operating scenario within a multi-agent system. At this state, we say that it has put forward its propositional attitude. It is still to state its intention. This is where ACL comes into play. An ACL is a set of primitives that allow an agent to state its intention. The primitives are the performatives that agent are permitted to use in an attempt to communicate with other agents.

In short, we can say that the ACL is the medium through which the intention regarding the content of the exchange between agents are communicated. Using such performatives as assert, request or query, with regards to content specified with the inner language.

Agent Communication Language

The increasing popularity of multi-agent based systems that required such complex interactions persuaded numerous efforts by various consortium to research into ACLs. Some better known examples

are the DARPA initiated KSE and FIPA's effort.

The KSE was set up with the aim of developing techniques and software tools to allow for efficient communication and co-ordination between agents that can be re-used and eventually become the common tool for all agents based systems.

The KSE concerned itself with the development of each of the three layers mentioned. The Interlingua group developed an inner language, that of Knowledge Interchange Format (KIF). This serves as a common language for expressing the content of a knowledge base.

The Shared, Reusable Knowledge Bases (SRKB) group simply provides a plethora of sharable ontologies and tools.

Acronym	Full Form
ACL	**Agent Communication Language**
DARPA	Defense Advanced Research Projects Agency
KSE	Knowledge Sharing Effort
FIPA	Foundation for Intelligent Physical Agent
KQML	Knowledge Query Manipulation Language

Table of acronyms

Finally and most importantly, the External Interface Group is the group responsible for churning out an ACL. The result of which was the emergence of Knowledge Query Manipulation Language (KQML). KQML is a message format that describes the structure of a message that is passed between agents in a run-time knowledge sharing system. It also provides a library of open-ended primitives or performatives that describe loosely the permissible actions/operations that agents may attempt in communicating with one another.

The FIPA is a non-profit organisation that was started to encourage and promote agent-based applications, services and equipment. FIPA is supported by a huge list of major industrial giants, such as NEC, Alcatel, NHK and Siemens. It consists of technical committees assigned to topical as well as long standing issues regarding agent-based systems. One of which is charged with the responsibility with developing an ACL.

The result of which, was the FIPA ACL. Which is an outer language that specifies message format and include descriptions of their pragmatics, that is the communicative acts or intentions of the agents.

Note that these ACLs are merely message formats as well as providing library after library performatives or "instructions" to describe an agent's intention. Multi-agent based systems often impl

ement subsets of the performatives and more often than not, dialects of the ACLs are developed. This is caused by the fact that the ACL does not have a fixed semantics. Setting fixed semantics would mean that programmer loses flexibility on designing autonomous and heterogeneous agents.

Conclusion

Despite many industry and non-developers adopting some variant of KQML, systems using different dialect of KQML still cannot inter-operate. There still lacks a universally agreed upon semantics foundation.

However, multi-agent based systems research is still immature, and further efforts by both FIPA and KSE are hoped to solve the impending issues. Furthermore, the development of KQML have played an important role in describing what an ACL is and what it should entail.

Article 2 would look into the details of the two ACLs, and a comparison would be made between the two along with examples of implementations of them.

References

Co-ordinating PLans of Autonomous Agents, F.Von Martial, In *Lecture Notes in AI,* Springer-Verlag, ISBN 3-540-55615-x

Introduction to Software Agents, Chpt. 1, In *Software Agents,* J. Bradshaw, MIT Press, ISBN 0-262-522349

KQML as ACL, T. Finin, Y. Labrou and Mayfield, Paper in *Software Agents,* J. Bradshaw, MIT Press, ISBN 0-262-522349

Communicative actions for artificial agents, P.Cohen, H. Levesque, In *proceedings of 1st International Conference on multi-agent systems*

Semantics and conversations for an ACL, Y.Labrou, T. Finin, Paper in *Readins in Agents,* Morgan-Kaufmann Publishers, ISBN 1-55860-495-2, M. Huhns, M. Singh

Agent Communication Language: Rethinking the Principle, In *IEEE Computer*, December 1998, pg.40-47, M.Singh

Agent Communication Language: The Current Landscape, In *IEEE Intelligent Systems,* March/April 1999, pg.45-52, Y.Labrou, T.Finin, Y.Peng

Webpage: *Agent Communication Language and Protocol* (http://agents.umbc.edu/technology/acl.shtml)

Webpage: *European Agent Link Homepage (http://www.agentlink.org)*

Webpage: *IC Online ACL-based Agent Systems* (http://www.computer.org/internet/v4n2/w2agents.htm)

KQML as an Agent Communication Language
(1994) (Make Corrections) (382 citations)
Tim Finin, Richard Fritzson, Don McKay, Robin McEntire
Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)

Abstract: This paper describes the design of and experimentation with the Knowledge Query and Manipulation Language (KQML), a new language and protocol for exchanging information and knowledge. This work is part of a larger effort, the ARPA Knowledge Sharing Effort which is aimed at developing techniques and methodology for building large-scale knowledge bases which are sharable and reusable. KQML is both a message format and a message-handling protocol to support run-time knowledge sharing among agents. ... (Update)

Context of citations to this paper: More

...ndcessaires h la communication entre les agents en plus de fournir un vdrificateur de la syntaxe de KQML. **Rappelons que KQML [14] est un langage d interrogation et de manipulation des connaissances.** C est un langage basd sur les acres du langage naturel [8] I1...

...and it s modification in an explicit way. **All the agents use Knowledge Query and Manipulation Language (KQML) for communication [8].** AP and PA multiagent system play a key role in enabling INTERLABS distributed virtual campus Each INTERLABS client uses a Web navigator...

Cited by: More
Engineering Infrastructures for Mobile Organizations - Cabri, Leonardi, Mamei.. (2001) (Correct)
Interaction Modal Logic for multiagent systems based - On Bdi Architecture (Correct)
Stream Oriented Interactions for Highly - Distributed And Disconnected (Correct)

Similar documents (at the sentence level): More
31.0%: KQML as an Agent Communication Language - Finin, Labrou, Mayfield (1994) (Correct)
27.4%: KQML as an agent communication language - Finin, Labrou, Mayfield (1995) (Correct)
19.1%: Desiderata for Agent Communication Languages - Mayfield, Labrou, Finin (1995) (Correct)

Active bibliography (related documents): More All
0.3: Cognitive Modeling and Group Adaptation in Intelligent.. - Leonardo Garrido (Correct)
0.3: Dynamic Agency: a Methodology and Architecture for Multiagent.. - Amigoni (2000) (Correct)
0.2: A Security Architecture for Agent Communication Languages - Mayfield, Finin (Correct)

Similar documents based on text: More All
0.8: An Overview of KQML: A Knowledge Query and.. - Chalupsky, Finin.. (1992) (Correct)
0.7: A Security Architecture Based on Trust Management for.. - Systems Lalana Kagal (2002) (Correct)
0.7: A Reactive Service Composition Architecture for.. - Chakraborty.. (2002) (Correct)

Related documents from co-citation: More All
17: Software Agents - Genesereth, Ketchpel - 1994
14: Enabling technology for knowledge sharing (context) - Neches, Fikes et al. - 1991
10: A Translation Approach to Portable Ontology Specifications (context) - Gruber - 1993

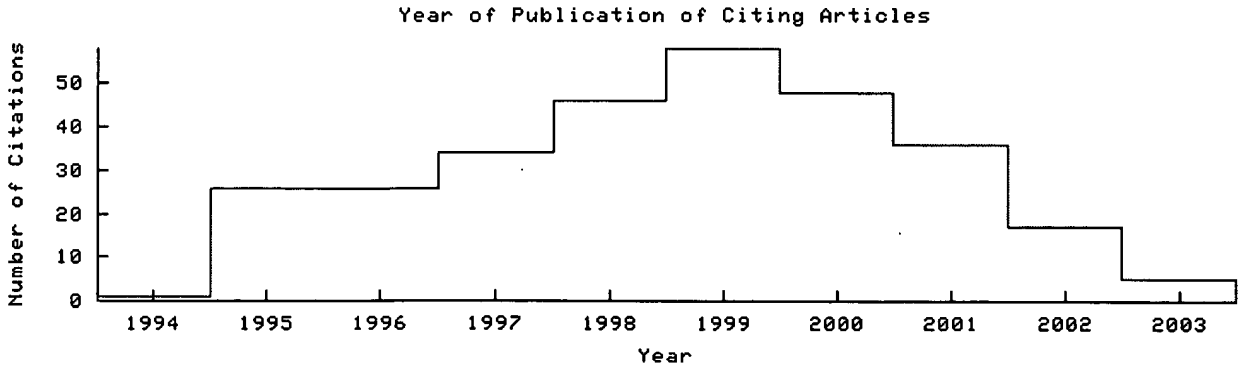
BibTeX entry: (Update)

T. Finin et.al.: "KQML as an Agent Communication Language ", 3rd International Conference on Information and Knowledge Management (CIKM94), ACM Press, December 1994
http://citeseer.ist.psu.edu/article/finin94kqml.html More

```
@inproceedings{ finin94kqml,
    author = "T. Finin and R. Fritzson and D. McKay and R. McEntire",
    title = "{KQML as an Agent Communication Language}",
    booktitle = "Proceedings of the 3rd International Conference on Information and
    publisher = "ACM Press",
    address = "Gaithersburg, MD, USA",
    editor = "N. Adam and B. Bhargava and Y. Yesha",
    pages = "456--463",
    year = "1994",
    url = "citeseer.ist.psu.edu/article/finin94kqml.html" }
```

Citations (may not include all citations):

187 Enabling technology for knowledge sharing (context) - Neches, Fikes et al. - 1991
179 Practical Planning: Extending the Classical AI Planning Para.. (context) - Wilkins - 1988
174 Knowledge interchange format (context) - Genesereth, Fikes - 1992
104 PACT: An experiment in integrating concurrent engineering sy.. - Cutkosky, Engelmore et al. - 1993
82 The DARPA knowledge sharing effort: Progress report - Patil, Fikes et al. - 1992
82 The DARPA knowledge sharing effort: Progress report - Patil, Fikes et al. - 1992
73 A semantics approach for KQML -- a general purpose communica.. - Labrou, Finin - 1994
66 The LOOM knowledge representation language (context) - MacGregor, Bates - 1987
50 Shade: Technology for knowledgebased collaborative - Kuokka - 1993
50 Shade: Technology for knowledgebased collaborative engineeri.. - McGuire - 1993
41 Planning and reformulating queries for semantically-modeled .. - Arens - 1992
38 Genesereth and Steven P (context) - Michael - 1994
33 An intelligent agent framework for enterprise integration (context) - Pan, Tenenbaum - 1991
22 Viewconcepts: Knowledge-based access to databases - Pastor, McKay et al. - 1992
21 Enterprise integration: Lessons from shade and pact (context) - Tenenbaum, Weber et al. - 1993
16 Knowledge interchange format: The KIF of death - Ginsberg - 1991
14 An agent-based approach to software interoperability (context) - Genesereth - 1993
11 and Stefano Ceri (context) - Wiederhold, Wegner - 1992
10 Query processing in an information mediator - Arens, Chee et al. - 1994
7 SOCAP: system for operations crisis action planning - Bienkowski, desJardins et al. - 1994
6 Specification of the KQML agentcommunication language (context) - Working, Knowledge et al. - 1992
5 The knowledge representation specification language manual (context) - Lehrer - 1994
3 A knowledge query and manipulation language for intelligent .. (context) - Finin, Fritzson et al. - 1992
3 Cosmos: A system for supporting design negotiation (context) - Mark - 1994
1 A highlevel language and protocol to support intelligent age.. (context) - Finin, Fritzson et al. - 1992
1 Expanding the Definition of Database (context) - Finin, Nicholas et al. - 1993



The graph only includes citing articles where the year of publication is known.

Documents on the same site (http://www.inf.ufsc.br/iad/users/c/carlos/eastman.htm):
Modelling Interaction in Agent Systems - Dalmonte, Gaspari (1995) (Correct)

Google Web Images Groups News Froogle[New!] more »
+"Agent Communication language" Search Advanced Search Preferences

Web Results **1** - **10** of about **10,400** for **+"Agent Communication language"**. (**0.38** seconds)

UMBC KQML Web
A page from the UMBC KQML Web -- a collection of web pages on the KQML **Agent Communication Language**. UMBC KQML Web UMBC LAIT | AgentWeb ...
www.cs.umbc.edu/kqml/ - 5k - Cached - Similar pages

[PS] KQML as an **Agent Communication Language**
File Format: Adobe PostScript - View as Text
KQML as an **Agent Communication Language**. \Lambda. Tim Finin and Richard Fritzson. Computer Science Department University of Maryland Baltimore County. ...
www.cs.umbc.edu/kqml/papers/kqml-acl.ps - Similar pages
[More results from www.cs.umbc.edu]

FIPA **Agent Communication Language** Specifications
... FIPA Agent Communication specifications deal with **Agent Communication Language** (ACL) messages, message exchange interaction protocols, speech act theory-based ...
www.fipa.org/repository/aclspecs.html - 31k - Cached - Similar pages

Agent Communication Language
... This will be the focus of this article, the wonderful world of **Agent Communication Language** or ACLs. ... **Agent Communication Language**. ...
infoeng.ee.ic.ac.uk/~malikz/surprise2001/ hsh99e/article1/ - 12k - Cached - Similar pages

Agent Communication language
Agent Communication language (ACL). ... Form: A good **agent communication language** should be declarative, syntactically simple and readable by people. ...
infoeng.ee.ic.ac.uk/~malikz/surprise2001/ kt197/article2/main.html - 46k - Cached - Similar pages

KQML as an **Agent Communication Language** - Finin (ResearchIndex)
KQML as an **Agent Communication Language** (1994) (Make Corrections) (382 citations) Tim Finin, Richard Fritzson, Don McKay, Robin McEntire. ...
citeseer.ist.psu.edu/20174.html - 15k - Cached - Similar pages

KQML as an **agent communication language** - Finin (ResearchIndex)
KQML as an **agent communication language** (1995) (Make Corrections) (382 citations) Tim Finin, Yannis Labrou, James Mayfield. Proceedings ...
citeseer.ist.psu.edu/26577.html - 25k - Cached - Similar pages
[More results from citeseer.ist.psu.edu]

Autonomous agent 2000 Workshop on Agent Communication
... A first attempt to come to a standardised **agent communication language** (ACL) came forth from the DARPA knowledge sharing project and produced KQML. ...
wwwis.win.tue.nl/ac2000/ - 35k - Cached - Similar pages

RoboCup-Rescue Simulation **Agent Communication Language**
Agent Communication Language for RoboCup-Rescue Simulation Prototype. Morita's Message. hi This is the design plan of the communication ...
www.rescuesystem.org/robocuprescue/acl.html - 26k - Cached - Similar pages

ACL, **Agent Communication Language**
Term. ACL, **Agent Communication Language**. Mentioned, ... Now with TMs. ACL KQML Agent Communication Languages. FIPA's **Agent Communication Language** (ACL) FIPA ACL. ...
www.infoloom.com/gcaconfs/WEB/ts1273/tp1273.HTM - 3k - Cached - Similar pages

Gooooooooogle ▶
Result Page: 1 2 3 4 5 6 7 8 9 10 **Next**

+"Agent Communication languag Search

Search within results | Language Tools | Search Tips | Dissatisfied? Help us improve

Google Home - Advertising Solutions - Business Solutions - About Google

FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS

FIPA ACL Message Structure Specification

Document title	FIPA ACL Message Structure Specification		
Document number	SC00061G	**Document source**	FIPA TC Communication
Document status	Standard	**Date of this status**	2002/12/03
Supersedes	None		
Contact	fab@fipa.org		
Change history	See *Informative Annex A — ChangeLog*		

http://www.fipa.org/
Geneva, Switzerland

Foreword

The Foundation for Intelligent Physical Agents (FIPA) is an international organization that is dedicated to promoting the industry of intelligent agents by openly developing specifications supporting interoperability among agents and agent-based applications. This occurs through open collaboration among its member organizations, which are companies and universities that are active in the field of agents. FIPA makes the results of its activities available to all interested parties and intends to contribute its results to the appropriate formal standards bodies where appropriate.

The members of FIPA are individually and collectively committed to open competition in the development of agent-based applications, services and equipment. Membership in FIPA is open to any corporation and individual firm, partnership, governmental body or international organization without restriction. In particular, members are not bound to implement or use specific agent-based standards, recommendations and FIPA specifications by virtue of their participation in FIPA.

The FIPA specifications are developed through direct involvement of the FIPA membership. The status of a specification can be either Preliminary, Experimental, Standard, Deprecated or Obsolete. More detail about the process of specification may be found in the FIPA Document Policy [f-out-00000] and the FIPA Specifications Policy [f-out-00003]. A complete overview of the FIPA specifications and their current status may be found on the FIPA Web site.

FIPA is a non-profit association registered in Geneva, Switzerland. As of June 2002, the 56 members of FIPA represented many countries worldwide. Further information about FIPA as an organization, membership information, FIPA specifications and upcoming meetings may be found on the FIPA Web site at http://www.fipa.org/.

Contents

1 Scope

This document contains specifications for the FIPA ACL message parameters. The objectives of standardizing the form of a FIPA-compliant ACL message are:

- To help ensure interoperability by providing a standard set of ACL message structure, and,
- To provide a well-defined process for maintaining this set.

2 FIPA ACL Message Structure

A FIPA ACL message contains a set of one or more message parameters. Precisely which parameters are needed for effective agent communication will vary according to the situation; the only parameter that is mandatory in all ACL messages is the `performative`, although it is expected that most ACL messages will also contain `sender`, `receiver` and `content` parameters.

If an agent does not recognize or is unable to process one or more of the parameters or parameter values, it can reply with the appropriate `not-understood` message.

Specific implementations are free to include user-defined message parameters other than the FIPA ACL message parameters specified in *Table 1*. The semantics of these user-defined parameters is not defined by FIPA, and FIPA compliance does not require any particular interpretation of these parameters. The prefatory string "`x-`" must be used for the names of these non-FIPA standard additional parameters.

Some parameters of the message might be omitted when their value can be deduced by the context of the conversation. However, FIPA does not specify any mechanism to handle such conditions, therefore those implementations that omit some message parameters are not guaranteed to interoperate with each other.

The full set of FIPA ACL message parameters is shown in *Table 1* without regard to their specific encodings in an implementation. FIPA-approved encodings and parameter orderings for ACL messages are given in other specifications. Each ACL message representation specification contains precise syntax descriptions for ACL message encodings based on XML, text strings and several other schemes.

A FIPA ACL message corresponds to the abstract parameter message payload identified in the [FIPA00001].

Parameter	Category of Parameters
`performative`	Type of communicative acts
`sender`	Participant in communication
`receiver`	Participant in communication
`reply-to`	Participant in communication
`content`	Content of message
`language`	Description of Content
`encoding`	Description of Content
`ontology`	Description of Content
`protocol`	Control of conversation
`conversation-id`	Control of conversation
`reply-with`	Control of conversation
`in-reply-to`	Control of conversation
`reply-by`	Control of conversation

Table 1: FIPA ACL Message Parameters

The following terms are used to define the ontology and the abstract syntax of the FIPA ACL message structure:

- **Frame**. This is the mandatory name of this entity that must be used to represent each instance of this class.
- **Ontology**. This is the name of the ontology, whose domain of discourse includes their parameters described in the table.

- **Parameter**. This identifies each component within the frame. The type of the parameter is defined relative to a particular encoding. Encoding specifications for ACL messages are given in their respective specifications.

- **Description**. This is a natural language description of the semantics of each parameter. Notes are included to clarify typical usage.

- **Reserved Values**. This is a list of FIPA-defined constants associated with each parameter. This list is typically defined in the specification referenced.

All of the FIPA message parameters share the frame and ontology shown in *Table 2*.

Frame	`fipa-acl-message`
Ontology	`fipa-acl`

Table 2: FIPA ACL Message Frame and Ontology

2.1 Type of Communicative Act

2.1.1 Performative

Parameter	Description	Reserved Values
`performative`	Denotes the type of the communicative act of the ACL message	See [FIPA00037]

Notes: The `performative` parameter is a required parameter of all ACL messages. Developers are encouraged to use the FIPA standard performatives (see [FIPA00037]) whenever possible.

2.2 Participants in Communication

2.2.1 Sender

Parameter	Description	Reserved Values
`sender`	Denotes the identity of the sender of the message, that is, the name of the agent of the communicative act.	

Notes: The `sender` parameter will be a parameter of most ACL messages. It is possible to omit the `sender` parameter if, for example, the agent sending the ACL message wishes to remain anonymous. The `sender` parameter refers to the agent which performs the communicative act giving rise to this ACL message.

2.2.2 Receiver

Parameter	Description	Reserved Values
`receiver`	Denotes the identity of the intended recipients of the message.	

Notes: Ordinarily, the `receiver` parameter will be a part of every ACL message. It is only permissible to omit the `receiver` parameter if the message recipient can be reliably inferred from context, or in special cases such as the embedded ACL message in `proxy` and `propagate`.

The `receiver` parameter may be a single agent name or a non-empty set of agent names. The latter corresponds to the situation where the message is multicast. Pragmatically, the semantics of this multicast is that the sender intends the message for each recipient of the CA encoded in the message. For example, if an agent performs an `inform` act with a set of three agents as receiver, it denotes that the sender intends each of these agents to come to believe the content of the message.

2.2.3 Reply To

Parameter	Description	Reserved Values
reply-to	This parameter indicates that subsequent messages in this conversation thread are to be directed to the agent named in the reply-to parameter, instead of to the agent named in the sender parameter.	

2.3 Content of Message

2.3.1 Content

Parameter	Description	Reserved Values
content	Denotes the content of the message; equivalently denotes the object of the action. The meaning of the content of any ACL message is intended to be interpreted by the receiver of the message. This is particularly relevant for instance when referring to referential expressions, whose interpretation might be different for the sender and the receiver.	

Notes: Most ACL messages require a content expression. Certain ACL message types, such as cancel, have an implicit content, especially in cases of using the conversation-id or in-reply-to parameters.

2.4 Description of Content

2.4.1 Language

Parameter	Description	Reserved Values
language	Denotes the language in which the content parameter is expressed.	See [FIPA00007]

Notes: The ACL content parameter is expressed in a formal language. This field may be omitted if the agent receiving the message can be assumed to know the language of the content expression.

2.4.2 Encoding

Parameter	Description	Reserved Values
encoding	Denotes the specific encoding of the content language expression.	See [FIPA00007]

Notes: The content expression might be encoded in several ways. The encoding parameter is optionally used to specify this encoding to the recipient agent. If the encoding parameter is not present, the encoding will be specified in the message envelope that encloses the ACL message.

2.4.3 Ontology

Parameter	Description	Reserved Values
ontology	Denotes the ontology(s) used to give a meaning to the symbols in the content expression.	

Notes: The ontology parameter is used in conjunction with the language parameter to support the interpretation of the content expression by the receiving agent. In many situations, the ontology parameter will be commonly understood by the agent community and so this message parameter may be omitted.

2.5 Control of Conversation

2.5.1 Protocol

Parameter	Description	Reserved Values
`protocol`	Denotes the interaction protocol that the sending agent is employing with this ACL message.	See [FIPA00025]

Notes: The `protocol` parameter defines the interaction protocol in which the ACL message is generated. This parameter is optional; however, developers are advised that employing ACL without the framework of an interaction protocol (and thus directly using the ACL semantics to control the agent's generation and interpretation of ACL messages) is an extremely ambitious undertaking.

Any ACL message that contains a non-null value for the `protocol` parameter is considered to belong to a conversation and it is required to respect the following rules:

- the initiator of the protocol must assign a non-null value to the `conversation-id` parameter,
- all responses to the message, within the scope of the same interaction protocol, should contain the same value for the `conversation-id` parameter, and,
- the timeout value in the `reply-by` parameter must denote the latest time by which the sending agent would like to have received the next message in the protocol flow (not be confused with the latest time by which the interaction protocol should terminate).

2.5.2 Conversation Identifier

Parameter	Description	Reserved Values
`conversation-id`	Introduces an expression (a conversation identifier) which is used to identify the ongoing sequence of communicative acts that together form a conversation.	

Notes: An agent may tag ACL messages with a conversation identifier to manage its communication strategies and activities. Typically this will allow an agent to identify individual conversations with multiple agents. It will also allow agents to reason across historical records of conversations.

It is required the usage of globally unique values for the `conversation-id` parameter in order to allow the participants to distinguish between several concurrent conversations. A simple mechanism to ensure uniqueness is the concatenation of the globally unique identifier of the sender agent to an identifier (for example, a progressive number) that is unique within the scope of the sender agent itself

2.5.3 Reply With

Parameter	Description	Reserved Values
`reply-with`	Introduces an expression that will be used by the responding agent to identify this message.	

Notes: The `reply-with` parameter is designed to be used to follow a conversation thread in a situation where multiple dialogues occur simultaneously. For example, if agent *i* sends to agent *j* a message which contains:

```
reply-with <expr>
```

Agent *j* will respond with a message containing:

```
in-reply-to <expr>
```

2.5.4 In Reply To

Parameter	Description	Reserved Values
`in-reply-to`	Denotes an expression that references an earlier action to which this message is a reply.	

Notes: See notes for Section 2.5.3.

2.5.5 Reply By

Parameter	Description	Reserved Values
`reply-by`	Denotes a time and/or date expression which indicates the latest time by which the sending agent would like to receive a reply.	

Notes: The time will be expressed according to the sender's view of the time on the sender's platform. The reply message can be identified in several ways: as the next sequential message in an interaction protocol, through the use of the `reply-with` parameter, through the use of a `conversation-id` and so forth. The way that the reply message is identified is determined by the agent implementer.

3 References

[FIPA00001] FIPA Abstract Architecture Specification. Foundation for Intelligent Physical Agents, 2000. http://www.fipa.org/specs/fipa00001/

[FIPA00007] FIPA Content Languages Library Specification. Foundation for Intelligent Physical Agents, 2000. http://www.fipa.org/specs/fipa00007/

[FIPA00025] FIPA Interaction Protocol Library Specification. Foundation for Intelligent Physical Agents, 2000. http://www.fipa.org/specs/fipa00025/

[FIPA00037] FIPA Communicative Act Library Specification. Foundation for Intelligent Physical Agents, 2000. http://www.fipa.org/specs/fipa00037/

4 Informative Annex A — ChangeLog

4.1 2002/10/01 - version F by TC X2S

Page 1, line 64: Removed references to maintenance procedures and inclusion criteria
Page 2, line 83: Added requirement that additional parameters have the "x-" prefix
Page 4, line 148: Specified that the content is intended to be interpreted by the receiver
Page 5, line 178: Added requirements to control the conversations
Page 5, line 184: Added requirement that `conversation-id` parameter be a globally unique identifier
Page 7, lines 222-260: Removed section 3 on maintenance of FIPA ACL

4.2 2002/12/03 - version G by FIPA Architecture Board

Entire document: Promoted to Standard status

Google Web Images Groups News Froogle[New!] more »
+"Agent Communication language" Search Advanced Search Preferences

Web Results **11** - **20** of about **10,400** for **+"Agent Communication language"**. (**0.28** seconds)

[PDF] TRENDS IN **AGENT COMMUNICATION LANGUAGE**
File Format: PDF/Adobe Acrobat - View as HTML
Computational Intelligence, Volume 2, Number 5, 2002 TRENDS IN **AGENT COMMUNICATION LANGUAGE** B. Chaib-draa* and F. Dignum** (*) Computer Science Department ...
www.damas.ift.ulaval.ca/~coursMAS/ chaib-draa02trends.pdf - Similar pages

Operational specification of a commitment-based **agent ...**
... Operational specification of a commitment-based **agent communication language**. Full text, pdf formatPdf (195 KB). Source, International ...
portal.acm.org/ citation.cfm?id=544868&dl=ACM&coll=portal&CFID=11111111&CFTOKEN=2222222 - Similar pages

Denotational semantics for **agent communication language**
... Denotational semantics for **agent communication language**. Full text, pdf formatPdf (268 KB). Source, International Conference on Autonomous ...
portal.acm.org/ citation.cfm?id=376427&dl=ACM&coll=portal&CFID=11111111&CFTOKEN=2222222 - Similar pages
[More results from portal.acm.org]

[PDF] TRENDS IN **AGENT COMMUNICATION LANGUAGE**
File Format: PDF/Adobe Acrobat
Computational Intelligence, Volume 18, Number 2, 2002 TRENDS IN **AGENT COMMUNICATION LANGUAGE** B. C HAIB - DRAA Laval University, Canada F. D IGNUM Utrecht ...
www.cs.uu.nl/people/dignum/papers/CIarticle1.pdf - Similar pages

KQML as an **agent communication language**
KQML as an **agent communication language**. Fachgebiet: Informationssysteme, ...
www.is.informatik.uni-duisburg.de/ bib/xml/Finin_etal_94.html - 6k - Cached - Similar pages

KQML as an **agent communication language**
KQML as an **agent communication language**. Working group: Information Systems, ...
www.is.informatik.uni-duisburg.de/ bib/xml/Finin_etal_94.html.en - 6k - Cached - Similar pages
[More results from www.is.informatik.uni-duisburg.de]

FACL: A Form-Based **Agent Communication Language** for Enduser ...
October 25 - 28, 2000 Taipei, Taiwan. p. 139 FACL: A Form-Based **Agent Communication Language** for Enduser-Initiative Agent-Based Application Development. PDF. ...
csdl.computer.org/comp/proceedings/compsac/ 2000/0792/00/07920139abs.htm - 12k - Cached - Similar pages

RoboCup-Rescue Simulation **Agent Communication Language** Version 0
Agent Communication Language for RoboCup-Rescue Simulation Prototype. Morita's Message. This is revised version of civilian language specification ver. ...
sakura.meijo-u.ac.jp/ttakaHP/kiyosu/ robocup/Rescue/acl0.html - 10k - Cached - Similar pages

FIPA 1997 part 2: **Agent Communication Language**
FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS. FIPA 97 Specification. Part 2. **Agent Communication Language**. Obsolete. Publication date: 10 th October, 1997. ...
www.fipa.org/specs/fipa00018/OC00018A.html - 101k - Cached - Similar pages
[More results from www.fipa.org]

May 1999 : XML-based Agent Communication: VPN Provisioning as a ...
... example). For example, 'an agent should speak an **agent communication language**', or 'an agent has beliefs, desires and intentions'. ...
www.infoloom.com/gcaconfs/WEB/granada99/bau.HTM - 38k - Cached - Similar pages
[More results from www.infoloom.com]

◀ Goooooooooooogle ▶
Result Page: **Previous** 1 **2** 3 4 5 6 7 8 9 10 11 **Next**

+"Agent Communication languag Search

Search within results | Language Tools | Search Tips

Google Home - Advertising Solutions - Business Solutions - About Google

FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS

FIPA 97 Specification

Part 2

Agent Communication Language

Obsolete

Publication date: 10th October, 1997

Geneva, Switzerland

Contents

Foreword

The Foundation for Intelligent Physical Agents (FIPA) is a non-profit association registered in Geneva, Switzerland. FIPA's purpose is to promote the success of emerging agent-based applications, services and equipment. This goal is pursued by making available in a timely manner, internationally agreed specifications that maximise interoperability across agent-based applications, services and equipment. This is realised through the open international collaboration of member organisations, which are companies and universities active in the agent field. FIPA intends to make the results of its activities available to all interested parties and to contribute the results of its activities to appropriate formal standards bodies.

This specification has been developed through direct involvement of the FIPA membership. The 35 corporate members of FIPA (October 1997) represent 12 countries from all over the world

Membership in FIPA is open to any corporation and individual firm, partnership, governmental body or international organisation without restriction. By joining FIPA each Member declares himself individually and collectively committed to open competition in the development of agent-based applications, services and equipment. Associate Member status is usually chosen by those entities who do want to be members of FIPA without using the right to influence the precise content of the specifications through voting.

The Members are not restricted in any way from designing, developing, marketing and/or procuring agent-based applications, services and equipment. Members are not bound to implement or use specific agent-based standards, recommendations and FIPA specifications by virtue of their participation in FIPA.

This specification is published as FIPA 97 ver. 1.0 after two previous versions have been subject to public comments following disclosure on the WWW. It has undergone intense review by members as well non-members. FIPA is now starting a validation phase by encouraging its members to carry out field trials that are based on this specification. During 1998 FIPA will publish FIPA 97 ver. 2.0 that will incorporate whatever adaptations will be deemed necessary to take into account the results of field trials.

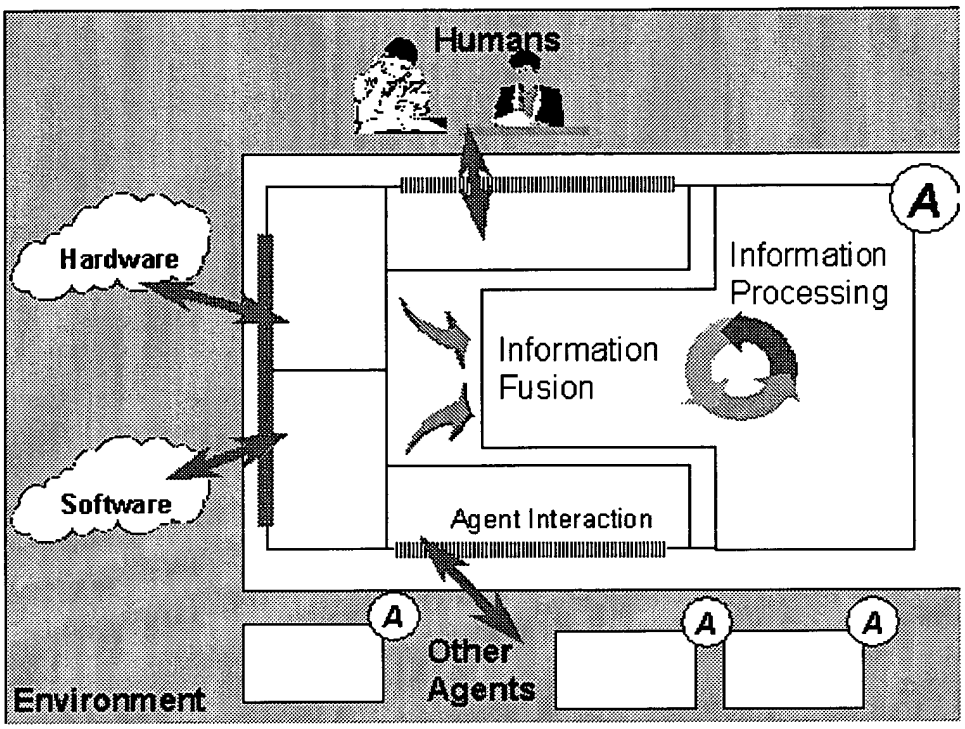
This document forms part two of the FIPA '97 specification. It should be read in conjunction with parts one (Agent Management) and three (Agent/Software Interaction). Part One, *Agent Management* details standards for agent naming, message transport mechanisms and possible failures, and agent registration. Part Three, *Agent/Software Integration* details how agent systems can inter-operate successfully with non-agent software systems, such as databases and legacy applications.

This release of part two of the FIPA 97 specification cancels and replaces all previous draft versions.

Introduction

The FIPA 97 specification is the first output of the Foundation for Intelligent Physical Agents. It provides specification of basic agent technologies that can be integrated by agent systems developers to make complex systems with a high degree of interoperability.

FIPA specifies the interfaces of the different components in the environment with which an agent can interact, i.e. humans, other agents, non-agent software and the physical world.



Summary of agent interactions with their environment

FIPA produces two kinds of specification:

— **normative** specifications that mandate the external behaviour of an agent and ensure interoperability with other FIPA-specified subsystems;

— **informative** specifications of applications for guidance to industry on the use of FIPA technologies.

The first set of specifications – called FIPA 97 – has seven parts:

— three normative parts for basic agent technologies: agent management, **agent communication language** and agent/software integration

— four informative application descriptions that provide examples of how the normative items can be applied: personal travel assistance, personal assistant, audio-visual entertainment and broadcasting and network management and provisioning.

Overall, the three FIPA 97 technologies allow:

— the construction and management of an agent system composed of different agents, possibly built by different developers;

— agents to communicate and interact with each other to achieve individual or common goals;

— legacy software or new non-agent software systems to be used by agents.

A brief summary of the FIPA 97 specification is given below.

Part 1 Agent Management

This part of FIPA 97 provides a normative framework within which FIPA compliant agents can exist, operate and be managed. It defines an agent platform reference model containing such capabilities as white and yellow pages, message routing and life-cycle management. True to the FIPA approach, these capablities are themselves intelligent agents using formally sound communicative acts based on special message sets. An appropriate ontology and content language allows agents to discover each other's capabilities.

Part 2 Agent Communication Language

The FIPA **Agent Communication Language** (ACL) is based on speech act theory: messages are actions, or *communicative acts*, as they are intended to perform some action by virtue of being sent. The specification consists of a set of message types and the description of their pragmatics, that is the effects on the mental attitudes of the sender and receiver agents. Every communicative act is described with both a narrative form and a formal semantics based on modal logic.

The specifications include guidance to users who are already familiar with KQML in order to facilitate migration to the FIPA ACL.

The specification also provides the normative description of a set of high-level interaction protocols, including requesting an action, contract net and several kinds of auctions.

Part 3 Agent/Software Integration

This part applies to any other non-agentised software with which agents need to "connect". Such software includes legacy software, conventional database systems, middleware for all manners of interaction including hardware drivers. Because in most significant applications, non-agentised software may dominate software agents, part 3 provides important normative statements. It suggests ways by which Agents may connect to software via "wrappers" including specifications of the wrapper ontology and the software dynamic registration mechanism. For this purpose, an Agent Resource Broker (ARB) service is defined which allows advertisement of non-agent services in the agent domain and management of their use by other agents, such as negotiation of parameters (e.g. cost and priority), authentication and permission.

Part 4 - Personal Travel Assistance

The travel industry involves many components such as content providers, brokers, and personalization services, typically from many different companies. In applying agents to this industry, various implementations from various vendors must interoperate and dynamically discover each other as different services come and go. Agents operating on behalf of their users can provide assistance in the pre-trip planning phase, as well as during the on-trip execution phase. A system supporting these services is called a PTA (Personal Travel Agent).

In order to accomplish this assistance, the PTA interacts with the user and with other agents, representing the available travel services. The agent system is responsible for the configuration and delivery - at the right time, cost, Quality of Service, and appropriate security and privacy measures - of trip planning and guidance services. It provides examples of agent technologies for both the hard requirements of travel such as airline, hotel, and car arrangements as well as the soft added-value services according to personal profiles, e.g. interests in sports, theatre, or other attractions and events.

Part 5 - Personal Assistant

One central class of intelligent agents is that of a personal assistant (PA). It is a software agent that acts semi-autonomously for and on behalf of a user, modelling the interests of the user and providing services to the user or other people and PAs as and when required. These services include managing a user's diary, filtering and sorting e-mail, managing the user's activities, locating and delivering (multimedia) information, and planning entertainment and travel. It is like a secretary, it accomplishes routine support tasks to allow the user to concentrate on the real job, it is unobtrusive but ready when needed, rich in knowledge about user and work. Some of the services may be provided by other agents (e.g. the PTA) or systems, the Personal Assistant acts as an interface between the user and these systems.

In the FIPA'97 test application, a Personal Assistant offers the user a unified, intelligent interface to the management of his personal meeting schedule. The PA is capable of setting up meetings with several participants, possibly involving travel for some of them. In this way FIPA is opening up a road for adding interoperability and agent capabilities to the already established

Part 6 - Audio/Video Entertainment & Broadcasting

An effective means of information filtering and retrieval, in particular for digital broadcasting networks, is of great importance because the selection and/or storage of one's favourite choice from plenty of programs on offer can be very impractical. The information should be provided in a customised manner, to better suit the user's personal preferences and the human interaction with the system should be as simple and intuitive as possible. Key functionalities such as profiling, filtering, retrieving, and interfacing can be made more effective and reliable by the use of agent technologies.

Overall, the application provides to the user an intelligent interface with new and improved functionalities for the negotiation, filtering, and retrieval of audio-visual information. This set of functionalities can be achieved by collaboration between a user agent and content/service provider agent.

Part 7 - Network management & provisioning

Across the world, numerous service providers emerge that combine service elements from different network providers in order to provide a single service to the end customer. The ultimate goal of all parties involved is to find the best deals available in terms of Quality of Service and cost. Intelligent Agent technology is promising in the sense that it will facilitate automatic negotiation of appropriate deals and configuration of services at different levels.

Part 7 of FIPA 97 utilizes agent technology to provide dynamic Virtual Private Network (VPN) services where a user wants to set up a multi-media connection with several other users.

The service is delivered to the end customer using co-operating and negotiating specialized agents. Three types of agents are used that represent the interests of the different parties involved:

— agents to communicate and interact with each other to achieve individual or common goals;

— The Service Provider Agent (SPA) that represents the interests of the Service Provider.

— The Network Provider Agent (NPA) that represents the interests of the Network Provider.

The service is established by the initiating user who requests the service from its PCA. The PCA negotiates in with available SPAs to obtain the best deal available. The SPA will in turn negotiate with the NPAs to obtain the optimal solution and to configure the service at network level. Both SPA and NPA communicate with underlying service- and network management systems to configure the underlying networks for the service.

Document history

This document is release 1.0 of part 2 of the FIPA 97 standard. Draft versions of this document have been reviewed within FIPA and by the agent community. Changes from previous draft versions are not recorded here. However, future revisions will be noted in this section.

1 Scope

"Language is a very difficult thing to put into words" – *Voltaire*

This document forms part two of the FIPA 97 specification for interoperable agents and agent societies. In particular, this document lays out underlying principles and detailed requirements for agents to be able to communicate with each other using messages representing communicative acts, independently of the specific agent implementations.

The document lays out, in the sections below, the following:

— A core set of communicative acts, their meaning and means of composition;

— Common patterns of usage of these communicative acts, including standard composite messages, and standard or commonly used interaction protocols;

— A detailed semantic description of the underlying meaning of the core set of message primitives;

— A summary of the relationship between the FIPA ACL and widely used *de facto* standard **agent communication language** KQML.

Objectives of this document

This document is intended to be directly of use to designers, developers and systems architects attempting to design, build and test agent applications, particularly communities of multiple agents. It aims to lay out clearly the practical components of inter-agent communication and co-operation, and explain the underlying theory. Beyond a basic appreciation of the model of agent communication, readers can make practical use of the ACL specification without necessarily absorbing the detail of the formal basis of the language.

However, the language does have a well-defined formal semantic foundation. The intention of this semantics is that it both gives a deeper understanding of the meaning of the language to the formally inclined, and provides an unambiguous reference point. This will be of increasing importance as agents, independently developed by separate individuals and teams, attempt to inter-operate successfully.

This part of the FIPA 97 specification defines a language and supporting tools, such as protocols, to be used by *intelligent software agents* to communicate with each other. The technology of software agents imposes a high-level view of such agents, deriving much of its inspiration from social interaction in other contexts, such as human-to-human communication. Therefore, the terms used and the mechanisms used support such a higher-level, often *task based*, view of interaction and communication. The specification does not attempt to define the low and intermediate level services often associated with communication between distributed software systems, such as network protocols, transport services, etc. Indeed, the existence of such services used to physically convey the byte sequences comprising the inter-agent communication acts are assumed.

No single, universal definition of a software agent exists, nor does this specification attempt to define one. However, some characteristics of agent behaviour are commonly adopted, and the communication language defined in this specification sets out to support and facilitate these behaviours. Such characteristics include, but are not limited to:

— Goal directed behaviour;

— Autonomous determination of courses of action;

— Interaction by negotiation and delegation;

— Modelling of anthropomorphic mental attitudes, such as beliefs, intentions, desires, plans and commitments;

— Flexibility in responding to situations and needs.

No expectation is held that any given agent will necessarily embody any or all of these characteristics. However, it is the intention of this part of the specification that such behaviours are supported by the communication language and its supporting framework where appropriate.

Note on conformance to the underlying semantic model

The semantic model described in this document is given solely as an informative reference point for agent behaviour, as there is currently no agreed technology for compliance testing against the semantics of the epistemic operators used in the model.

This is due to the difficulty of verifying that the mental attitudes of an agent conform to the specification, without dictating the agent's internal architecture or underlying implementation model. As such, the semantics cannot be considered normative until the issue of compliance testing is resolved. Such tests will be the subject of further FIPA work.

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this specification are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid specifications.

ISO/IEC 2022: *Information technology - Character code.*

FIPA 97 specification – Part 1: Agent Management.

FIPA 97 specification – Part 3: Agent/Software Integration.

3 Terms and definitions

For the purposes of this specification, the following terms and definitions apply:

Action

A basic construct which represents some activity which an agent may perform. A special class of actions is the communicative acts.

ARB Agent

An agent which provides the Agent Resource Broker (ARB) service. There must be at least one such an agent in each Agent Platform in order to allow the sharing of non-agent services.

Agent

An Agent is the fundamental actor in a domain. It combines one or more service capabilities into a unified and integrated execution model which can include access to external software, human users and communication facilities.

Agent Communication Language (ACL)

A language with precisely defined syntax, semantics and pragmatics that is the basis of communication between independently designed and developed software agents. ACL is the primary subject of this part of the FIPA specification.

Agent Communication Channel (ACC) Router

The Agent Communication Channel is an agent which uses information provided by the Agent Management System to route messages between agents within the platform and to agents resident on other platforms.

Agent Management System (AMS)

The Agent Management System is an agent which manages the creation, deletion, suspension, resumption, authentication and migration of agents on the agent platform and provides a "white pages" directory service for all agents resident on an agent platform. It stores the mapping between globally unique agent names (or GUID) and local transport addresses used by the platform.

Agent Platform (AP)

An Agent Platform provides an infrastructure in which agents can be deployed. An agent must be registered on a platform in order to interact with other agents on that platform or indeed other platforms. An AP consists of three capability sets ACC, AMS and default Directory Facilitator.

Communicative Act (CA)

A special class of actions that correspond to the basic building blocks of dialogue between agents. A communicative act has a well-defined, declarative meaning independent of the content of any given act. CA's are modelled on speech act theory. Pragmatically, CA's are performed by an agent sending a message to another agent, using the message format described in this specification.

Content

That part of a communicative act which represents the domain dependent component of the communication. Note that "the content of a message" does not refer to "everything within the message, including the delimiters", as it does in some languages, but rather specifically to the domain specific component. In the ACL semantic model, a content expression may be composed from propositions, actions or IRE's.

Conversation

An ongoing sequence of communicative acts exchanged between two (or more) agents relating to some ongoing topic of discourse. A conversation may (perhaps implicitly) accumulate context which is used to determine the meaning of later messages in the conversation.

Software System

A software entity which is not conformant to the FIPA Agent Management specification.

CORBA:

Common Object Request Broker Architecture, an established standard allowing object-oriented distributed systems to communicate through the remote invocation of object methods.

Directory Facilitator (DF)

The Directory facilitator is an agent which provides a "yellow pages" directory service for the agents. It store descriptions of the agents and the services they offer.

Feasibility Precondition (FP)

The conditions (i.e. one or more propositions) which need be true before an agent can (plan to) execute an action.

Illocutionary effect

See speech act theory.

Knowledge Querying and Manipulation Language (KQML)

A de facto (but widely used) specification of a language for inter-agent communication. In practice, several implementations and variations exist.

Local Agent Platform

The Local Agent Platform is the AP to which an aget is attached and which represents an ultimate destination for messages directed to that agent.

Message

An individual unit of communication between two or more agents. A message corresponds to a communicative act, in the sense that a message encodes the communicative act for reliable transmission between agents. Note that communicative acts can be recursively composed, so while the outermost act is directly encoded by the message, taken as a whole a given message may represent multiple individual communicative acts.

Message content

See content.

Message transport service

The message transport service is an abstract service provided by the agent management platform to which the agent is (currently) attached. The message transport service provides for the reliable and timely delivery of messages to their destination agents, and also provides a mapping from agent logical names to physical transport addresses.

Ontology

An ontology gives meanings to symbols and expressions within a given domain language. In order for a message from one agent to be properly understood by another, the agents must ascribe the same meaning to the constants used in the message. The ontology performs the function of mapping a given constant to some well-understood meaning. For a given domain, the ontology may be an explicit construct or implicitly encoded with the implementation of the agent.

Ontology sharing problem

The problem of ensuring that two agents who wish to converse do, in fact, share a common ontology for the domain of discourse. Minimally, agents should be able to discover whether or not they share a mutual understanding of the domain constants. Some research work is addressing the problem of dynamically updating agents' ontologies as the need arises. This specification makes no provision for dynamically sharing or updating ontologies.

Perlocutionary Effect

See speech act theory.

Proposition

A statement which can be either true or false. A closed proposition is one which contains no variables, other than those defined within the scope of a quantifier.

Protocol

A common pattern of conversations used to perform some generally useful task. The protocol is often used to facilitate a simplification of the computational machinery needed to support a given dialogue task between two agents. Throughout this document, we reserve protocol to refer to dialogue patterns between agents, and networking protocol to refer to underlying transport mechanisms such as TCP/IP.

Rational Effect (RE)

The rational effect of an action is a representation of the effect that an agent can expect to occur as a result of the action being performed. In particular, the rational effect of a communicative act is the perlocutionary effect an agent can expect the CA to have on a recipient agent.

Note that the recipient is not bound to ensure that the expected effect comes about; indeed it may be impossible for it to do so. Thus an agent may use its knowledge of the rational effect in order to plan an action, but it is not entitled to believe that the rational effect necessarily holds having performed the act.

Speech Act Theory

A theory of communications which is used as the basis for ACL. Speech act theory is derived from the linguistic analysis of human communication. It is based on the idea that with language the speaker not only makes statements, but also performs actions. A speech act can be put in a stylised form that begins "I hereby request ..." or "I hereby declare ...". In this form the verb is called the performative, since saying it makes it so. Verbs that cannot be put into this form are not speech acts, for example "I hereby solve this equation" does not actually solve the equation. [Austin 62, Searle 69].

In speech act theory, communicative acts are decomposed into locutionary, illocutionary and perlocutionary acts. Locutionary acts refers to the formulation of an utterance, illocutionary refers to a categorisation of the utterance from the speakers perspective (e.g. question, command, query, etc), and perlocutionary refers to the other intended effects on the hearer. In the case of the ACL, the perlocutionary effect refers to the updating of the agent's mental attitudes.

Software Service

An instantiation of a connection to a software system.

TCP/IP

A networking protocol used to establish connections and transmit data between hosts

Wrapper Agent

An agent which provides the FIPA-WRAPPER service to an agent domain on the Internet.

4 Symbols (and abbreviated terms)

ACC: Agent Communication Channel

ACL: **Agent Communication Language**

AMS: Agent Management System

AP: Agent Platform

API: Application Programming Interface

ARB: Agent Resource Broker

CA: Communicative Act

CORBA: Common Object Request Broker Architecture

DCOM:	Distributed COM
DF:	Directory Facilitator
FIPA:	Foundation for Intelligent Physical Agents
FP:	Feasibility Precondition
GUID:	Global Unique Identifier
HAP:	Home Agent Platform
HTTP:	Hypertext Transmission Protocol
IDL:	Interface Definition Language
IIOP:	Internet Inter-ORB Protocol
OMG:	Object Management Group
ORB:	Object Request Broker
RE:	Rational Effect
RMI:	Remote Method Invocation, an inter-process communication method embodied in Java
SL:	Semantic Language
SMTP:	Simple Mail Transfer Protocol
TCP / IP:	Transmission Control Protocol / Internet Protocol

5 Overview of Inter-Agent Communication

5.1 Introduction

This specification document does not define in a precise, prescriptive way what an agent is nor how it should be implemented. Besides the lack of a general consensus on this issue in the agent research community, such definitions frequently fall into the trap of being overly restrictive, ruling out some software constructs whose developers legitimately consider to be agents, or else overly weak and of little assistance to the reader or software developer. A goal of this specification is to be as widely applicable as possible, so the stance taken is to define the components as precisely as possible, and allow applicability in any particular instance to be decided by the reader.

Nevertheless, some position must be taken on some of the characteristics of an agent, that it, on what an agent can *do*, in order that the specification can specify a means of doing it. This position is outlined here, and consists of an *abstract characterisation* of agent properties, and a simple abstract model of inter-agent communication.

The first characteristic assumed is that agents are communicating at a higher level of discourse, i.e. that the contents of the communication are meaningful statements about the agents' environment or knowledge. This is one characteristic that differentiates agent communication from, for example, other interactions between strongly encapsulated computational entities such as method invocation in CORBA.

In order for this discourse to be given meaning, some assumptions have to be made about the agents. In this specification, an abstract characterisation of agents is assumed, in which some core capabilities of agents are described in terms of the agent's *mental attitudes*. This characterisation or model is intended as an abstract specification, i.e. it does not pre-determine any particular agent implementation model nor a cognitive architecture.

More specifically, this specification characterises an agent as being able to be described as though it has mental attitudes of:

- **Belief**, which denotes the set of propositions (statements which can be true or false) which the agent accepts are (currently) true; propositions which are believed false are represented by believing the negation of the proposition.
- **Uncertainty**, which denotes the set of propositions which the agent accepts are (currently) not known to be certainly true or false, but which are held to be more likely to be true than false; propositions which are uncertain but more likely to be false are represented by being uncertain of the negation of the proposition. Note that this attitude does not prevent an agent from adopting a specific uncertain information formalism, such as probability theory, in which a proposition is believed to have a certain degree of support. Rather the uncertainty attitude provides a least commitment mechanism for agents with differing representation schemes to discuss uncertain information.
- **Intention**, which denotes a *choice*, or property or set of properties of the world which the agent desires to be true and which are not currently believed to be true. An agent which adopts an intention will form a plan of action to bring about the state of the world indicated by its choice.

Note that, with respect to some given proposition *p*, the attitudes of believing *p*, believing *not p*, being uncertain of *p* and being uncertain of *not p* are mutually exclusive.

In addition, agents understand and are able to perform certain *actions*. In a distributed system, an agent typically will only be able to fulfil its intentions by influencing other agents to perform actions.

Influencing the actions of other agents is performed by a special class of actions, denoted *communicative acts*. A communicative act is performed by one agent towards another. The mechanism of performing a communicative act is precisely that of sending a message encoding the act. Hence the roles of initiator and recipient of the communicative act are frequently denoted as the *sender* and *receiver* of the message, respectively.

Building from a well-defined core, the messages defined in this specification represent a set of communicative acts that attempt to seek a balance between generality, expressive power and simplicity, together with perspicuity to the agent developer. The message type defines the communicative action that is being performed. Together with the appropriate domain knowledge, the communicative act allows the receiver to determine the meaning of the contents of the message.

The meanings of the communicative acts given in §6.5 are given in terms of the pre-conditions in respect to the sender's mental attitudes, and the expected (from the sender's point of view) consequences on the receiver's mental attitudes. However, since the sender and receiver are independent, there is no guarantee that the expected consequences come to pass. For example, agent *i* may believe that "it is better to read books than to watch TV", and may intend *j* to come to believe so also. Agent i will, in the ACL, *inform j* of its belief in the truth of that statement. Agent *j* will then know, from the semantics of *inform*, that *i* intends it to believe in the value of books, but whether *j* comes itself to believe the proposition is a matter for *j* alone to decide.

This specification concerns itself with inter-agent communication through message passing. Key sections of the discussion are as follows:

- §5.2 discusses the transportation of messages between agents;
- §6.3 introduces the structure of messages;
- §6.4 gives a standard transport syntax for transmitting ACL messages over simple byte streams;
- §6.5 catalogues the standardised communicative acts and their representation as messages;
- §7 introduces and defines a set of communication protocols to simplify certain common sequences of messages;
- §8 formally defines the underlying communication model.

5.2 Message Transport Mechanisms

For two agents to communicate with each other by exchanging messages, they must have some common meeting point through which the messages are delivered. The existence and properties of this *message transport service* are the remit of FIPA Technical Committee 1: Agent Management.

The ACL presented here takes as a position that the contribution of agent technology to complex system behaviour and inter-operation is most powerfully expressed at what, for the lack of a better term, may be called the higher levels of interaction. For example, this document describes communicative acts for informing about believed truths, requesting complex actions, protocols for negotiation, etc. The interaction mechanisms presented here do not compete with, nor should they be compared to, low-level networking protocols such as TCP/IP, the OSI seven layer model, etc. Nor do they directly present an alternative to CORBA, Java RMI or Unix RPC mechanisms. However, the functionality of ACL does, in many ways overlap with the foregoing examples, not least in that ACL messages may often be expected to be delivered via such mechanisms.

The ACL's role may be further clarified by consideration of the FIPA goal of general open agent systems. Other mechanisms, notably CORBA, share this goal, but do so by imposing certain restrictions on the interfaces exposed by objects. History suggests that agents and agent systems are typically implemented with a greater variety of interface mechanisms; existing example agents include those using TCP/IP sockets, HTTP, SMTP and GSM short messages. ACL respects this diversity by attempting to minimise requirements on the message delivery service. Notably, the minimal message transport mechanism is defined as a textual form delivered over a simple byte stream, which is also the approach taken by the widely used KQML **agent communication language**. A potential penalty for this inclusive approach is upon very high-performance systems, where message throughput is pre-eminent. Future versions of this specification may define alternative transport mechanism assumptions, including other transport syntaxes, which meet the needs of very high performance systems.

Currently, the ACL imposes a minimal set of requirements on the message transport service, as shown below:

a) The message service is able to deliver a message, encoded in the transport form below, to a destination as a sequence of bytes. The message service exposes through its interface whether it is able to cope reliably with 8-bit bytes whose high-order bit may be set.

b) The normal case is that the message service is *reliable* (well-formed messages will arrive at the destination) *accurate* (the message is received in the form in which it was sent), and *orderly* (messages from agent a to agent b arrive at b in the order in which they were sent from a[1]). Unless informed otherwise, an agent is entitled to assume that these properties hold.

c) If the message delivery service is unable to guarantee any or all of the above properties, this fact is exposed in some way through the interface to the message delivery service

d) An agent will have the option of selecting whether it suspends and waits for the result of a message

(synchronous processing) or continues with other unrelated tasks while waiting for a message reply (asynchronous processing). The availability of this behaviour will be implementation specific, but it must be made explicit where either behaviour is not supported.

e) Parameters of *the act of delivering a message*, such as time-out if no reply, are not codified at the message level but are part of the interface exposed by the message delivery service.

f) The message delivery service will detect and report error conditions, such as: ill-formed message, undeliverable, unreachable agent, etc., back to the sending agent. Depending on the error condition, this may be returned either as a return value from the message sending interface, or through the delivery of an appropriate error message.

g) An agent has a name which will allow the message delivery service to deliver the message to the correct destination. The message delivery service will be able to determine the correct transport mechanism (TCP/IP, SMTP, http, etc.), and will allow for changes in agent location, as necessary.

The agent will, in some implementation specific way, have an structure which corresponds to a message it wishes to send or has received. The syntax shown below in this document defines a *transport form*, in which the message is mapped from its internal form to a character sequence, and can be mapped back to the internal message form from a given character sequence. Note again the absence of architectural commitment: the internal message form *may* be a explicit data structure, or it *may* be implicit in the way that the agent handles its messages.

For the purposes of the transport services, the message may be assumed to be an opaque byte stream, with the exception that it is possible to extract the destination of the message.

At this transport level, messages are assumed to be encoded in 7-bit characters according to the ISO/IEC 2022 standard. This specification allows the expression of characters in extended character sets, such as Japanese. The FIPA specification adopts the position that the default character mapping is US ASCII. More specifically, all ACL compliant agents should assume that, when communication is commenced:

— ISO/IEC 646 (US ASCII) is designated to G0;

— ISO/IEC 6429 C0 is designated;

— G0 is invoked in GL;

— C0 is invoked in CL;

— SPACE in 2/0 (0x20) and

— DELETE in 7/15 (0x7f)

Some transport services will be able to transport 8-bit characters safely, and, where this service is available, the agent is free to make use of it. However, safe transmission of 8-bit characters is not universally assumed.

6 FIPA ACL Messages

6.1 Preamble

This section defines the individual message types that are central to the ACL specification. In particular, the form of the messages and meaning of the message types are defined. The message types are a reference to the semantic acts defined in this specification. These types impart a meaning to the whole message, that is, the act and the content of the message, which extends any intrinsic meaning that the content itself may have.

For example, if *i* informs *j* that "Bonn is in Germany", the content of the message from *i* to *j* is "Bonn is in Germany", and the act is the act of *informing*. "Bonn is in Germany" has a certain meaning, and is true under any reasonable interpretation of the symbols "Bonn" and "Germany", but the meaning of the message includes effects on (the mental attitudes of) agents *i* and *j*. The determination of this effect is essentially a private matter to both i and j, but for meaningful communication to take place, some reasonable expectations of those effects must be fulfilled.

Clearly, the content of a message may range over an unrestricted range of domains. This specification does not mandate any one formalism for representing message content. Agents themselves must arrange to be able to interpret any given message content correctly. Note that this version of the specification does not address the *ontology sharing problem*, though future versions may do so. The specification does set out to specify the meanings of the acts independently of the content, that is, extending the above example, what it means to inform or be informed. In particular, a set of standard communicative acts and their meanings is defined.

It may be noted, however, that there is a trade-off between the power and specificity of the acts. Notionally, a large number of very specific act types, which convey nuances of meaning, can be considered equivalent to a smaller number of more general ones, but they place different representational and implementation constraints on the agents. The goals of the set of acts presented here are (i) to cover, overall, a wide range of communication situations, (ii) not to overtax the design of simpler agents intended to fulfil a specific, well-defined purpose, and (iii) to minimise redundancy and ambiguity, to facilitate the agent to choose which communicative act to employ. Succinctly, the goals are: completeness, simplicity and conciseness.

The fundamental view of messages in ACL is that a message represents a *communicative act*. For purposes of elegance and coherency, the treatment of communicative acts during dialogue should be consistent with the treatment of other actions; a given communicative action is just one of the actions that an agent can perform. The term *message* then plays two distinct roles within this document, depending on context. *Message* can be a synonym for *communicative act*, or it may refer to the computational structure used by the message delivery service to convey the agent's utterance to its destination.

The communication language presented in this specification is based on a precise formal semantics, giving an unambiguous meaning to communicative actions. In practice, this formal basis is supplemented with pragmatic extensions that serve to ease the practical implementation of effective inter-agent communications. For this reason, the message *parameters* defined below are not defined in the formal semantics in §8, but are defined in narrative form in the sections below. Similarly, conventions that agents are expected to adopt, such as protocol of message exchange, are given an operational semantics in narrative form only.

6.2 Requirements on agents

This document introduces a set of pre-defined message types and protocols that are available for all agents to use. However, it is not required for all agents to implement all of these messages. In particular, the minimal requirements on FIPA ACL compliant agents are as follows:

Requirement 1: Agents should send *not-understood* if they receive a message that they do not recognise or they are unable to process the content of the message. Agents must be prepared to receive and properly handle a *not-understood* message from other agents.
Requirement 2: An ACL compliant agent may choose to implement any subset (including all, though this is unlikely) of the pre-defined message types and protocols. The implementation of these messages must be correct with respect to the referenced act's semantic definition.
Requirement 3:

An ACL compliant agent which uses the communicative acts whose names are defined in this specification must implement them correctly with respect to their definition.
Requirement 4: Agents may use communicative acts with other names, not defined in this document, and are responsible for ensuring that the receiving agent will understand the meaning of the act. However, agents should not define new acts with a meaning that matches a pre-defined standard act.
Requirement 5: An ACL compliant agent must be able to correctly generate a syntactically well formed message in the transport form that corresponds to the message it wishes to send. Symmetrically, it must be able to translate a character sequence that is well-formed in the transport syntax to the corresponding message.

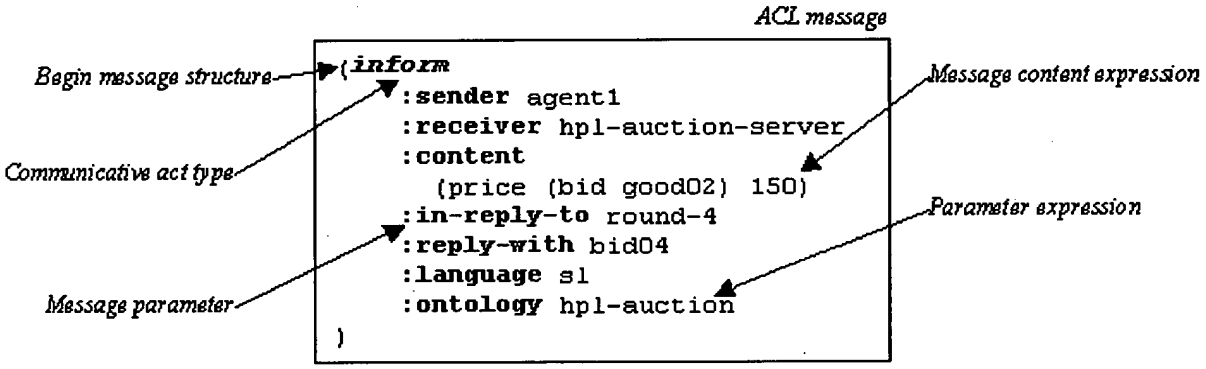
6.3 Message structure

This section introduces the various structural elements of a message.

6.3.1 Overview of ACL messages

The following figure summarises the main structural elements of an ACL message:

Figure 1 — Components of a message



In their transport form, messages are represented as s-expressions. The first element of the message is a word which identifies the communicative act being communicated, which defines the principal meaning of the message. There then follows a sequence of message parameters, introduced by parameter keywords beginning with a colon character. No space appears between the colon and the parameter keyword. One of the parameters contains the content of the message, encoded as an expression in some formalism (see below). Other parameters help the message transport service to deliver the message correctly (e.g. sender and receiver), help the receiver to interpret the meaning of the message (e.g. language and ontology), or help the receiver to respond co-operatively (e.g. reply-with, reply-by).

It is this transport form that is serialised as a byte stream and transmitted by the message transport service. The receiving agent is then responsible for decoding the byte stream, parsing the components message and processing it correctly.

Note that the message's communicative act type corresponds to that which in KQML is called the *performative*[2]).

6.3.2 Message parameters

As noted above, the message contains a set of one or more parameters. Parameters may occur in any order in the message. The only parameter that is mandatory in all messages is the *:receiver* parameter, so that the message delivery service can correctly deliver the message. Clearly, no useful message will contain only the receiver. However, precisely which other parameters are needed for effective communication will vary according to the situation.

The full set of pre-defined message parameters is shown in the following table:

Table 1 — Pre-defined message parameters

Message Parameter:	Meaning:
:sender	Denotes the identity of the sender of the message, i.e. the name of the agent of the communicative act.
:receiver	Denotes the identity of the intended recipient of the message. Note that the recipient may be a single agent name, or a tuple of agent names. This corresponds to the action of multicasting the message. Pragmatically, the semantics of this multicast is that the message is sent to each agent named in the tuple, and that the sender intends each of them to be recipient of the CA encoded in the message. For example, if an agent performs an inform act with a tuple of three agents as receiver, it denotes that the sender intends each of these agent to come to believe the content of the message.
:content	Denotes the content of the message; equivalently denotes the object of the action.

:reply-with	Introduces an *expression* which will be used by the agent responding to this message to identify the original message. Can be used to follow a conversation thread in a situation where multiple dialogues occur simultaneously.
	
	E.g. if agent i sends to agent j a message which contains
	`:reply-with query1,`
	agent j will respond with a message containing
	`:in-reply-to query1.`

:in-reply-to	Denotes an expression that references an earlier action to which this message is a reply.
:envelope	Denotes an expression that provides useful information about the message as seen by the message transport service. The content of this parameter is not defined in the specification, but may include time sent, time received, route, etc. The structure of the envelope is a list of keyword value pairs, each of which denotes some aspect of the message service.
:language	Denotes the encoding scheme of the content of the action.
:ontology	Denotes the ontology which is used to give a meaning to the symbols in the content expression.
:reply-by	Denotes a time and/or date expression which indicates a guideline on the latest time by which the sending agent would like a reply.
:protocol	Introduces an identifier which denotes the *protocol* which the sending agent is employing. The protocol serves to give additional context for the interpretation of the message. Protocols are discussed in §7.
:conversation-id	Introduces an expression which is used to identify an ongoing sequence of communicative acts which

	together form a conversation. A conversation may be used by an agent to manage its communication strategies and activities. In addition the conversation may provide additional context for the interpretation of the meaning of a message.

6.3.3 Message content

The *content* of a message refers to whatever the communicative act applies to. If, in general terms, the communicative act is considered as a sentence, the content is the grammatical object of the sentence. In general, the content can be encoded in any language, and that language will be denoted by the `:language` parameter. The only requirement on the content language is that it has the following properties:

Requirement 6:
In general, a content language must be able to express propositions, objects and actions. No other properties are required, though any given content language may be much more expressive than this. More specifically, the content of a message must express the data type of the action: propositions for inform, actions for request, etc.

— A *proposition* states that some sentence in a language is true or false. An *object*, in this context, is a construct which represents an identifiable "thing" (which may be abstract or concrete) in the domain of discourse. Object in this context does not necessarily refer to the specialised programming constructs that appear in *object-oriented* languages like C++ and Java. An *action* is a construct that the agent will interpret as being an activity which can be carried out by some agent. In general, an action does not produce a result which is communicated to another agent (but see, for example, § `(iota <variable> <term>)`
The iota operator introduces a scope for the given *expression* (which denotes a term), in which the given *identifier*, which would otherwise be free, is defined. An expression containing a free variable is not a well-formed SL expression. The expression "(iota x (P x)" may be read as "the x such that P [is true] of x. The *iota* operator is a constructor for terms which denote objects in the domain of discourse.

B.2.5).

Except in the special case outlined below, there is no requirement that message content languages conform to any well known (pre-defined) syntax. In other words, it is the *responsibility of the agents in a dialogue* to ensure that they are using a mutually comprehensible content language. This FIPA specification does not mandate the use of any particular content language. One suggested content language formalism is shown in Annex B. There are many ways to ensure the use of a common content language. It may be arranged by convention (e.g. such-and-such agents are well known always to use Prolog), by negotiation[3] among the parties, or by employing the services of an intermediary as a translator. Similarly, the agents are responsible for ensuring that they are using a common ontology.

The most general case is that of negotiating (i.e. jointly deciding) a content language. However, the agent must overcome the problem of being able to begin the conversation in the first place, in order that they can then negotiate content language. There has to be a common point of reference, known in advance to both parties. Thus, for the specific purpose of registering with a directory facilitator and performing other key agent management functions, the specification does include the following content language definition:

Definition 1:
The FIPA specification agent management content language is an s-expression notation used to express the propositions, objects and actions pertaining to the management of the agent's lifecycle. The terms in the expression are defined operationally in part one of the FIPA 97 specification.

Requirement 7:
A compliant agent is required to exercise the standard agent management capabilities through the use of messages using the agent management content language and ontology. The language and ontology are each denoted by the reserved term `fipa-agent-management` in their respective parameters.

6.3.4 Representing the content of messages

As noted above, the content of a message refers to the domain expression which the communicative act refers to. It is encoded in the message as the value of the *:content* parameter. The FIPA specification does not mandate any particular content encoding language (i.e. the representation form of the :content expression) on a normative basis. The SL content language is provided in Annex B on an informative basis.

To facilitate the encoding of simple languages (that is, simple in their syntactic requirements), the s-expression form included in the ACL grammar shown below allows the construction of s-expressions of arbitrary depth and complexity. A language which is defined as a sub-grammar of the general s-expression grammar is therefore admissible as a legal ACL message without modification. The SL grammar shown in Annex B is an example of exactly this approach.

However, agents commonly need to embed in the body of the message an expression encoded in a notation other than the simple s-expression form used for the messages themselves. The ACL grammar provides two mechanisms, both of which avoid the problem of an ACL parser being required to parse any expression in any language:

— Wrap the expression in double quotes, thus making it a string in ACL syntax, and protect any embedded double quote in the embedded expression with a backslash. Note that backslash characters in the content expression must also be protected. E.g.:

```
(inform :content "owner( agent1, \"Ian\" ) "
        :language Prolog
 ... )
```

— Prefix the expression with the appropriate length encoded string notation, thus ensuring that the expression will be treated as one lexical token irrespective of its structure. E.g.:

```
(inform :content #22"owner( agent1, "Ian" )
        :language Prolog
 ... )
```

As a result, an ACL parser will generate one lexical token, a string, representing the entire embedded language expression. Once the message has been parsed, the token representing the content expression can be interpreted according to its encoding scheme, which will by then be known from the *:language* parameter.

6.3.5 Use of MIME for additional content expression encoding

Sometimes, even the mechanisms in the previous section are not flexible enough to represent the full range of types of expression available to an agent. An example may be when an agent wishes to express a concept such as "the sound you asked for is <a digitised sound>". Alternatively, it may wish to express some content in a language or character set encoding different from that used for the description of the content, such as "the translation of error message <some English text> into Japanese is <some Japanese text>".

The Multipurpose Internet Mail Extensions (MIME) standard was developed to address similar issues in the context of Internet mail messages [Freed & Borenstein 96]. The syntactic form of MIME headers is suited particularly to the format of mail messages, and is not congruent with the transport syntax defined for FIPA ACL messages. However, the capabilities provided by MIME, and in particular the now widely used notation for annotating content types is a capability of great value to some categories of agent. To allow for this, an agent may optionally be able to process MIME content expression descriptions as wrappers around a given expression, using an extension of the ACL message syntax.

It is not a mandatory part of this specification that all agents be able to process MIME content descriptions. However, MIME-capable agents can register this ability with their directory facilitator, and then proceed to use the format defined in Annex D.

Note that, for the specific task of encoding language specific character sets, the ISO 2022 standard which is the base level character encoding of the message stream is capable of supporting a full range of international character encodings.

6.3.6 Primitive and composite communicative acts

This document defines a set of predefined communicative acts, each of which is given a specific meaning in the

specification. Pragmatically, each of these communicative acts may be treated equivalently: they have equal status. However, in terms of definition and the determination of the formal meaning of the communicative acts, we distinguish two classes: *primitive acts* and *composite acts.*

Primitive communicative acts are those whose actions are defined atomically, i.e. they are not defined in terms of other acts. Composite communicative acts are the converse. Acts are *composed* by one of the following methods:

— making one communicative act the object of another. For example, "I *request* you to *inform* me whether it is raining" is the composite *query-if* act.

— using the *composition operator* ";" to sequence actions

— using the composition operator "|" to denote a non-deterministic choice of actions.

The sequencing operator is written as an infix semicolon. Thus the expression:

```
a ; b
```

denotes an action, whose meaning is that of action a followed by action b.

The non-deterministic choice operator is written as an infix vertical bar. Thus the expression:

```
a | b
```

denotes a *macro action*, whose meaning is that of either action `a`, or action `b`, but not both. The action may occur in the past, present or future, or not at all.

Note that a macro action must be treated slightly differently than other communicative acts. A macro action can be planned by an agent, and requested by one agent of another. However, a macro act will not appear as the outermost (i.e. top-level) message being sent from one agent to another. Macro acts are used in the definition of new composite communicative acts. For example, see the *inform-if* act in §6.5.10.

The definition of composite actions in this way is part of the underlying semantic model for the ACL, defined using the semantic description language SL. Action composition as described above is not part of the concrete syntax for ACL. However, these operators are defined in the concrete syntax for SL used as a content language in Annex B.

6.4 Message syntax

This section defines the message transport syntax. The syntax is expressed in standard EBNF format. For completeness, the notation is as follows:

Grammar rule component	**Example**
Terminal tokens are enclosed in double quotes	`"("`
Non terminals are written as capitalised identifiers	`Expression`
Square brackets denote an optional construct	`[ "," OptionalArg ]`
Vertical bar denotes an alternative	`Integer | Real`
Asterisk denotes zero or more repetitions of the preceding expression	`Digit *`
Plus denotes one or more repetitions of the preceding expression	`Alpha +`
Parentheses are used to group expansions.	`( A | B ) *`
Productions are written with the non-terminal name on the lhs, expansion on the rhs, and terminated by a full stop.	`ANonTerminal = "an expansion".`

Some slightly different rules apply for the generation of lexical tokens. Lexical tokens use the same notation as above, except:

Lexical rule component	**Example**

Square brackets enclose a character set	`["a", "b", "c"]`
Dash in a character set denotes a range	`["a" - "z"]`
Tilde denotes the complement of a character set if it is the first character	`[ ~ "(", ")"]`
Post-fix question-mark operator denotes that the preceding lexical expression is optional (may appear zero or one times)	`["0" - "9"]? ["0" - "9"]`

6.4.1 Grammar rules for ACL message syntax

This section defines the grammar for ACL.

```
ACLCommunicativeAct = Message.

Message             = "(" MessageType MessageParameter* ")".

MessageType         = "accept-proposal"
                    | "agree"
                    | "cancel"
                    | "cfp"
                    | "confirm"
                    | "disconfirm"
                    | "failure"
                    | "inform"
                    | "inform-if"
                    | "inform-ref"
                    | "not-understood"
                    | "propose"
                    | "query-if"
                    | "query-ref"
                    | "refuse"
                    | "reject-proposal"
                    | "request"
                    | "request-when"
                    | "request-whenever"
                    | "subscribe".

MessageParameter    = ":sender" AgentName
                    | ":receiver" RecipientExpr
                    | ":content" ( Expression | MIMEEnhancedExpression )
                    | ":reply-with" Expression
                    | ":reply-by" DateTimeToken
                    | ":in-reply-to" Expression
                    | ":envelope" KeyValuePairList
                    | ":language" Expression
                    | ":ontology" Expression
                    | ":protocol" Word
                    | ":conversation-id" Expression.

Expression          = Word
                    | String
                    | Number
                    | "(" Expression * ")".

MIMEEnhancedExpression - optional extension. See Annex D.

KeyValuePairList    = "(" KeyValuePair * ")".
```

```
KeyValuePair       = "(" Word Expression ")".

RecipientExpr      = AgentName
                   | "(" AgentName + ")".

AgentName          = Word
                   | Word "@" AgentAddress.
AgentAddress       = Word "://" InternetAddress ":" Number "/" ACCObj
AccObj             = Word.

InternetAddress    = DNSName | IPAddress.
IPAddress          = Integer "." Integer "." Integer "." Integer.
DNSName            = Word.
```

Lexical rules

```
Word                      = [~ "\0x00" - "\0x1f",
                              "(", ")", "#", "0"-"9", "-"]
                            [~ "\0x00" - "\0x1f",
                              "(", ")"] *.

String                    = StringLiteral
                          | ByteLengthEncodedString.

StringLiteral             = "\""
                            ( [~ "\"" ] | "\\\"" )*
                            "\"".

ByteLengthEncodedString = "#" ["0" - "9"]+ "\""
                            <byte sequence>.

Number                    = Integer | Float.

DateTimeToken             = "+" ?
                            Year Month Day "T"
                            Hour Minute Second MilliSecond
                            (TypeDesignator ?).

Year                      = Digit Digit Digit Digit.
Month                     = Digit Digit.
Day                       = Digit Digit.
Hour                      = Digit Digit.
Minute                    = Digit Digit.
Second                    = Digit Digit.
MilliSecond               = Digit Digit Digit.
TypeDesignator            = AlphaCharacter.

Digit                     = ["0" - "9"].
```

6.4.2 Notes on grammar rules

a) The standard definitions for integers and floating point numbers are assumed.

b) All keywords are case-insensitive.

c) A length encoded string is a context sensitive lexical token. Its meaning is as follows: the header of the token is everything from the leading "#" to the separator " inclusive. Between the markers of the header is a decimal number with at least one digit. This digit then determines that *exactly* that number of 8-bit bytes are to be consumed as part of the token, without restriction. It is a lexical error for less than that number of bytes to be available.

Note that not all implementations of the agent communication channel (ACC) [see Part One of the FIPA 97 specification] will support the transparent transmission of 8-bit characters. It is the responsibility of the agent to ensure, by reference to the API provided for the ACC, that a given channel is able to faithfully transmit the chosen message encoding.

d) A well-formed message will obey the grammar, and in addition, will have at most one of each of the parameters. It is an error to attempt to send a message which is not well formed. Further rules on well-formed messages may be stated or implied the operational definitions of the values of parameters as these are further developed.

e) Strings encoded in accordance with ISO/IEC 2022 may contain characters which are otherwise not permitted in the definition of `Word`. These characters are ESC (0x1B), SO (0x0E) and SI (0x0F). This is due to the complexity that would result from including the full ISO/IEC 2022 grammar in the above EBNF description. Hence, despite the basic description above, a word may contain any well-formed ISO/IEC 2022 encoded character, other (representations of) parentheses, spaces, or the "#" character. Note that parentheses may legitimately occur as *part* of a well formed escape sequence; the preceding restriction on characters in a word refers only to the encoded characters, not the form of the encoding.

f) Time tokens are based on the ISO 8601 format, with extensions for relative time and millisecond durations. Time expressions may be absolute, or relative to the current time. Relative times are distinguished by the character "+" appearing as the first character in the construct. If no type designator is given, the local timezone is used. The type designator for UTC is the character "Z". UTC is preferred to prevent timezone ambiguities. Note that years must be encoded in four digits. As examples, 8:30 am on April 15th 1996 local time would be encoded as:

19960415T083000000

the same time in UTC would be:

19960415T083000000Z

while one hour, 15 minutes and 35 milliseconds from now would be:

+00000000T011500035.

g) The format defined for agent names is taken from part one of the FIPA 97 standard. The option of simply using a word as the agent name is only valid where that word can be unambiguously resolved to an full agent name in the format given.

6.5 Catalogue of Communicative Acts

This section defines all of the communicative acts that are part of this specification. Each message is defined by an informal narrative in this section, and more formally in §8. The narrative and formal definitions are intended to be equivalent. However, in the case of an ambiguity or inconsistency, the formal definition is the final reference point.

The following communicative acts and macro acts are standard components of the FIPA **agent communication language**. They are listed in alphabetical order. Communicative acts can be directly performed, can be planned by an agent, and can be requested of one agent by another. Macro acts can be planned and requested, but not directly performed.

6.5.1 Preliminary notes

The meanings of the communicative acts below frequently make reference to mental attitudes, such as belief, intention or uncertainty. Whilst the formal semantics makes reference to formal operators which express these concepts, a given agent implementation is not required to encode them explicitly, or to be founded on any particular agent model (e.g. BDI). In the following narrative definitions:

— *belief* means that, at least, the agent has a reasonable basis for stating the truth of a proposition, such as having the proposition stored in a data structure or expressed implicitly in the construction of the agent software;

— *intention* means that the agent wishes some proposition, not currently believed to be true, to become true, and further that it will act in such a way that the truth of the proposition will be established. Again, this may not be represented explicitly in the agent[4];

— *uncertain* means that the agent is not sure that a proposition is necessarily true, but it is more likely to be true than false. Believing a proposition and being uncertain of a proposition are mutually exclusive.

For ease of reference, a synopsis formal description of each act is included with the narrative text. The meaning of the notation used may be found in §8.

6.5.1.1 Category Index

The following table identifies the communicative acts in the catalogue by category. This is provided purely for ease of reference. Full descriptions of the messages can be found in the appropriate sections.

Table 2 — Categories of communicative acts

Communicative act	Information passing	Requesting information	Negotiation	Action performing	Error handling
accept-proposal			✓		
agree				✓	
cancel				✓	
cfp			✓		
confirm	✓				
disconfirm	✓				
failure					✓
inform	✓				
inform-if (macro act)	✓				
inform-ref (macro act)	✓				
not-understood					✓
propose			✓		
query-if		✓			
query-ref		✓			
refuse				✓	
reject-proposal			✓		
request				✓	
request-when				✓	
request-whenever				✓	
subscribe		✓			

6.5.2 accept-proposal

Summary:	The action of accepting a previously submitted proposal to perform an action.
Message content:	A tuple, consisting of an action expression denoting the action to be done, and a proposition giving the conditions of the agreement.
Description:	*Accept-proposal* is a general-purpose acceptance of a proposal that was previously submitted (typically through a *propose* act). The agent sending the acceptance informs the receiver that it intends that (at some point in the future) the receiving agent will perform the action, once the given precondition is, or becomes, true. The proposition given as part of the acceptance indicates the preconditions that the agent is attaching to the acceptance. A typical use of this is to finalise the details of a deal in some protocol. For example, a previous offer to "hold a meeting anytime on Tuesday" might be accepted with an additional condition that the time of the meeting is 11.00. Note for future extension: i may intend that a becomes done without necessarily
Summary Formal Model	$\langle i, \text{accept-proposal}(j, \langle j, a\rangle, p(e, \langle j, a\rangle))\rangle \equiv$ $\langle i, \text{inform}(j, I_i \text{Done}(\langle j, a\rangle, p(e, \langle j, a\rangle)))\rangle$ *Note: this summary is included here for completeness. For full details, see §8.*

Example	Agent i informs j that it accepts, without further preconditions, an offer from j to stream a given multimedia title to channel 19:
	
	`(accept-proposal`
	`  :sender i`
	`  :receiver j`
	`  :in-reply-to bid089`
	`  :content`
	`  (`
	`  (action j (stream-content movie1234 19))`
	`  true`
	`  )`
	`  :language sl)`
	
	Agent i informs j that it accepts an offer from j to stream a given multimedia title to channel 19 when the customer is ready. Agent i will *inform* j of this fact when appropriate:
	
	`(accept-proposal`
	`  :sender i`
	`  :receiver j`
	`  :in-reply-to bid089`
	`  :content`
	`  (`
	`  (action j (stream-content movie1234 19))`
	`  (B j (ready customer78))`
	`  )`
	`  :language sl)`

6.5.3 agree

Summary:	The action of agreeing to perform some action, possibly in the future.
Message content:	A tuple, consisting of an agent identifier, an action expression denoting the action to be done, and a proposition giving the conditions of the agreement.
Description:	*Agree* is a general purpose agreement to a previously submitted *request* to perform some action. The agent sending the agreement informs the receiver that it does intend to perform the action, but not until the given precondition is true. The proposition given as part of the *agree* act indicates the qualifiers, if any, that the agent is attaching to the agreement. This might be used, for example, to inform the receiver when the agent will execute the action which it is agreeing to perform. *Pragmatic note:* the precondition on the action being agreed to can include the perlocutionary effect of some other CA, such as an *inform* act. When the recipient of the agreement (e.g. a contract manager) wants the agreed action to be performed, it should then bring about the precondition by performing the necessary CA. This mechanism can be used to ensure that the contractor defers performing the action until the manager is ready for the action to be done.
Summary Formal Model	$\langle i, \text{agree}(j, a, p)\rangle =$ $\langle i, \text{Inform}(j, \text{I}_j \text{Done}(a, p))\rangle$ *Note: this summary is included here for completeness. For full details, see §8.*

Example	Agent i (a job-shop scheduler) requests j (a robot) to deliver a box to a certain location. J answers that it agrees to the request but it has low priority.
	
	`(request`
	`  :sender i`
	`  :receiver j`
	`  :content (action j (deliver box017 (location 12 19)))`
	`  :protocol fipa-request`
	`  :reply-with order567`
	`)`
	
	`(agree`
	`  :sender j`
	`  :receiver i`
	`  :content ((deliver j box017 (location 12 19))`
	`  (priority order567 low))`
	`  :in-reply-to order567`
	`  :protocol fipa-request`
	`)`

6.5.4 cancel

Summary:	The action of cancelling some previously *request*ed action which has temporal extent (i.e. is not instantaneous).
Message content:	An action expression denoting the action to be cancelled.
Description:	Cancel allows an agent to stop another agent from continuing to perform (or expecting to perform) an action which was previously requested. Note that the action that is the object of the act of cancellation should be believed by the sender to be ongoing or to be planned but not yet executed. Attempting to *cancel* an action that has already been performed will result in a *refuse* message being sent back to the originator of the request.
Summary Formal Model	$\langle i, \text{cancel}(j, a)\rangle \equiv$ $\langle i, \text{disconfirm}(j, I_j\ \text{Done}(a)\rangle$ *Note: this summary is included here for completeness. For full details, see §8.*

Example	Agent j0 asks i to cancel a previous *request-whenever* by quoting the action:
	`(cancel`
	`  :sender j0`
	`  :receiver i`
	`  :content (request-whenever :sender j ...)`
	`)`
	Agent j1 asks i to cancel an action by cross-referencing the previous conversation in which the request was made:
	`(cancel`
	`  :sender j1`
	`  :receiver i`
	`  :conversation-id cnv0087`
	`)`

6.5.5 cfp

Summary:	The action of calling for proposals to perform a given action.
Message content:	A tuple containing an action expression denoting the action to be done and a proposition denoting the preconditions on the action.
Description:	*CFP* is a general-purpose action to initiate a negotiation process by making a call for proposals to perform the given action. The actual protocol under which the negotiation process is established is known either by prior agreement, or is explicitly stated in the *:protocol* parameter of the message. In normal usage, the agent responding to a *cfp* should answer with a proposition giving its conditions on the performance of the action. The responder's conditions should be compatible with the conditions originally contained in the *cfp*. For example, the *cfp* might seek proposals for a journey from Frankfurt to Munich, with a condition that the mode of travel is by train. A compatible proposal in reply would be for the 10.45 express train. An incompatible proposal would be to travel by 'plane. Note that *cfp* can also be used to simply check the availability of an agent to perform some action.
Summary Formal Model	$\langle i, \text{cfp}(j, \langle j, a\rangle, p(e, \langle j, a\rangle))\rangle \equiv$ $\langle i, \text{query-ref}(j, \iota x$ $(I_j \forall e\ \text{Feasible}(e, \text{Done}(e; \langle i, \text{inform}(j, I_i \langle j, a\rangle)\rangle) \wedge$ $((x = p'(e, \langle j, a\rangle)) \wedge x \wedge p(e, \langle j, a\rangle)$ $\Rightarrow$ $I_j \text{Done}(\langle j, a\rangle) \wedge \text{Feasible}(\langle j, a\rangle)))))\rangle$ *Note: this summary is included here for completeness. For full details, see §8.*

Example	Agent j asks i to submit its proposal to sell 50 boxes of plums:
	
	`(cfp`
	`  :sender j`
	`  :receiver i`
	`  :content ((action i (sell plum 50)) true)`
	`  :ontology fruit-market`
	`)`

6.5.6 confirm

Summary:	The sender informs the receiver that a given proposition is true, where the receiver is known to be uncertain about the proposition.
Message content:	A proposition
Description:	The sending agent: • believes that some proposition is true • intends that the receiving agent also comes to believe that the proposition is true • believes that the receiver is *uncertain* of the truth of the proposition The first two properties defined above are straightforward: the sending agent is sincere [5], and has (somehow) generated the intention that the receiver should know the proposition (perhaps it has been asked). The last pre-condition determines when the agent should use *confirm* vs. *inform* vs. *disconfirm*: *confirm* is used precisely when the other agent is already known to be uncertain about the proposition (rather than *uncertain* about the negation of the proposition). From the receiver's viewpoint, receiving a *confirm* message entitles it to believe that: • the sender believes the proposition that is the content of the message • the sender wishes the receiver to believe that proposition also. Whether or not the receiver does, indeed, change its mental attitude to one of belief in the proposition will be a function of the receiver's trust in the sincerity and reliability of the sender.
Summary Formal Model	$\langle i, \text{confirm}(j, \phi)\rangle$ FP: $B_i\phi \wedge B_i U_j \phi$ RE: $B_j\phi$ *Note: this summary is included here for completeness. For full details, see §8.*

Examples	Agent i confirms to agent j that it is, in fact, true that it is snowing today.
	`(confirm`
	`:sender i`
	`:receiver j`
	`:content "weather( today, snowing )"`
	`:language Prolog)`

6.5.7 disconfirm

Summary:	The sender informs the receiver that a given proposition is false, where the receiver is known to believe, or believe it likely that, the proposition is true.
Message content:	A proposition
Description:	The disconfirm act is used when the agent wishes to alter the known mental attitude of another agent. The sending agent: • believes that some proposition is false • intends that the receiving agent also comes to believe that the proposition is false • believes that the receiver either believes the proposition, or is *uncertain* of the proposition. The first two properties defined above are straightforward: the sending agent is sincere *(note 5)*, and has (somehow) generated the intention that the receiver should know the proposition (perhaps it has been asked). The last pre-condition determines when the agent should use *confirm*, *inform* or *disconfirm*: *disconfirm* is used precisely when the other agent is already known to believe the proposition or to be uncertain about it. From the receiver's viewpoint, receiving a *disconfirm* message entitles it to believe that: • the sender believes that the proposition that is the content of the message is false; • the sender wishes the receiver to believe the negated proposition also. Whether or not the receiver does, indeed, change its mental attitude to one of disbelief in the proposition will be a function of the receiver's trust in the sincerity and reliability of the sender.
Summary Formal Model	$\langle i$, disconfirm(j, ϕ)$\rangle$ FP: $B_i \neg\phi \wedge B_i(U_j\phi \vee B_j\phi)$ RE: $B_j \neg\phi$ *Note: this summary is included here for completeness. For full details, see §8.*

Examples	Agent i, believing that agent j thinks that a shark is a mammal, attempts to change j's belief:
	`(disconfirm`
	`:sender i`
	`:receiver j`
	`:content (mammal shark))`

6.5.8 failure

Summary:	The action of telling another agent that an action was attempted but the attempt failed.
Message content:	A tuple, consisting of an action expression and a proposition giving the reason for the failure.
Description:	The failure act is an abbreviation for informing that an act was considered feasible by the sender, but was not completed for some given reason. The agent receiving a failure act is entitled to believe that: • the action has not been done • the action is (or, at the time the agent attempted to perform the action, was) feasible The (causal) reason for the refusal is represented by the proposition, which is the third term of the tuple. It may be the constant *true*. There is no guarantee that the reason is represented in a way that the receiving agent will understand: it could be a textual error message. Often it is the case that there is little either agent can do to further the attempt to perform the action.
Summary Formal Model	$\langle i, \text{failure}(j, a, p)\rangle \equiv$ $\langle i, \text{inform}(j, (\exists e)\ \text{Single}(e) \wedge e \neq a \wedge \text{Done}(e, \text{Feasible}(a) \wedge I_i\ \text{Done}(a)) \wedge$ $p \wedge (\neg\text{Done}(a) \wedge \neg I_i\ \text{Done}(a))\rangle$ *Note: this summary is included here for completeness. For full details, see §8.*

Example	Agent j informs i that it has failed to open a file:
	`(failure`
	`:sender j`
	`:receiver i`
	`:content`
	`(`
	`(action j "open( \"foo.txt\" )")`
	`(error-message "No such file: foo.txt")`
	`)`
	`:language sl)`

6.5.9 inform

Summary:	The sender informs the receiver that a given proposition is true.
Message content:	A proposition
Description:	The sending agent: • holds that some proposition is true; • intends that the receiving agent also comes to believe that the proposition is true; • does not already believe that the receiver has any knowledge of the truth of the proposition. The first two properties defined above are straightforward: the sending agent is sincere, and has (somehow) generated the intention that the receiver should know the proposition (perhaps it has been asked). The last property is concerned with the semantic soundness of the act. If an agent knows already that some state of the world holds (that the receiver knows proposition *p*), it cannot rationally adopt an intention to bring about that state of the world (i.e. that the receiver *comes to know p* as a result of the inform act). Note that the property is not as strong as it perhaps appears. The sender *is not* required to *establish* whether the receiver knows p. It is only the case that, in the case that the sender already happens to know about the state of the receiver's beliefs, it should not adopt an intention to tell the receiver something it already knows. From the receiver's viewpoint, receiving an inform message entitles it to believe that: • the sender believes the proposition that is the content of the message • the sender wishes the receiver to believe that proposition also. Whether or not the receiver does, indeed, adopt belief in the proposition will be a function of the receiver's trust in the sincerity and reliability of the sender.
Summary Formal Model	$\langle i, \text{inform}(j, \phi)\rangle$ FP: $B_i\phi \wedge \neg B_i(Bif_j\phi \vee Uif_j\phi)$ RE: $B_j\phi$ *Note: this summary is included here for completeness. For full details, see §8.*

Examples	Agent i informs agent j that (it is true that) it is raining today:
	`(inform`
	`:sender i`
	`:receiver j`
	`:content "weather( today, raining )"`
	`:language Prolog)`

6.5.10 inform-if (macro act)

Summary:	A macro action for the agent of the action to inform the recipient whether or not a proposition is true.
Message content:	A proposition.
Description:	The *inform-if* macro act is an abbreviation for informing whether or not a given proposition is believed. The agent which enacts an *inform-if* macro-act will actually perform a standard *inform* act. The content of the inform act will depend on the informing agent's beliefs. To *inform-if* on some closed proposition ϕ: • if the agent believes the proposition, it will inform the other agent that ϕ • if it believes the negation of the proposition, it informs that ϕ is false (i.e. $\neg\phi$) Under other circumstances, it may not be possible for the agent to perform this plan. For example, if it has no knowledge of ϕ, or will not permit the other party to know (that it believes) ϕ, it will send a *refuse* message.
Summary Formal Model	$\langle i,$ inform-if(j, p)$\rangle \equiv$ $\langle i,$ inform(j, p)$\rangle \mid \langle i,$ inform(j, $\neg$p)$\rangle$ *Note: this summary is included here for completeness. For full details, see §8.*

Examples	Agent i requests j to inform it whether Lannion is in Normandy:
	`(request`
	`  :sender i`
	`  :receiver j`
	`  :content`
	`  (inform-if :sender j`
	`  :receiver i`
	`  :content "in( lannion, normandy )"`
	`  :language Prolog)`
	`  :language sl)`
	Agent j replies that it is not:
	`(inform :sender j`
	`  :receiver i`
	`  :content "\+ in( lannion, normandy )"`
	`  :language Prolog)`

6.5.11 inform-ref (macro act)

Summary:	A macro action for sender to inform the receiver the object which corresponds to a definite descriptor (e.g. a name).
Message content:	An object description.
Description:	The *inform-ref* macro action allows the sender to inform the receiver some object that the sender believes corresponds to a definite descriptor, such as a name or other identifying description. *Inform-ref* is a macro action, since it corresponds to a (possibly infinite) disjunction of *inform* acts, each of which informs the receiver that "the object corresponding to *name* is *x*" for some given x. For example, an agent can plan an *inform-ref* of the current time to agent j, and then perform the act "*inform* j that the time is 10.45". The agent performing the act should believe that the object corresponding to the definite descriptor is the one that is given, and should not believe that the recipient of the act already knows this. The agent may elect to send a *refuse* message if it is unable to establish the preconditions of the act. Alternatively, it may choose to alter another agents known mental attitudes with respect to the given description by *confirm-ref* or *disconfirm-ref.*
Summary Formal Model	$\langle i,$ inform-ref($j, \iota x\ \delta(x)$)$\rangle \equiv$ $\langle i,$ inform($j, \iota x\ \delta(x) = r_1$)$\rangle$ | ... | $\langle i,$ inform($j, \iota x\ \delta(x) = r_k$)$\rangle$ *Note: this summary is included here for completeness. For full details, see §8.*

Example	Agent i requests j to tell it the current Prime Minister of the United Kingdom:
	
	`(request`
	`  :sender i`
	`  :receiver j`
	`  :content`
	`    (inform-ref`
	`      :sender j`
	`      :receiver i`
	`      :content (iota ?x (UKPrimeMinister ?x))`
	`      :ontology world-politics`
	`      :language sl`
	`    )`
	`  :reply-with query0`
	`  :language sl)`
	
	Agent j replies:
	
	`(inform`
	`    :sender j`
	`    :receiver i`
	`    :content (= (iota ?x (UKPrimeMinister ?x))`
	`                "Tony Blair")`
	`    :ontology world-politics`
	`    :in-reply-to query0)`
	
	Note that a standard abbreviation for the *request* of *inform-ref* used in this example is the act *query-ref.*

6.5.12 not-understood

Summary:	The sender of the act (e.g. i) informs the receiver (e.g. j) that it perceived that j performed some action, but that i did not understand what j just did. A particular common case is that i tells j that i did not understand the message that j has just sent to i.
Message content:	A tuple consisting of an action or event (e.g. a communicative act) and an explanatory reason.
Description:	The sender received a communicative act which it did not understand. There may be several reasons for this: the agent may not have been designed to process a certain act or class of acts, or it may have been expecting a different message. For example, it may have been strictly following a pre-defined protocol, in which the possible message sequences are predetermined. The *not-understood* message indicates to that the sender of the original (i.e. misunderstood) action that nothing has been done as a result of the message. This act may also be used in the general case for i to inform j that it has not understood j's action. The second term of the content tuple is a proposition representing the reason for the failure to understand. There is no guarantee that the reason is represented in a way that the receiving agent will understand: it could be a textual error message. However, a co-operative agent will attempt to explain the misunderstanding constructively
Summary Formal Model	<*i*, not-understood(*j*, a)> ≡ FP: to be completed RE: to be completed *Note: this summary is included here for completeness. For full details, see §8.*

Examples	Agent i did not understand an query-if message because it did not recognise the ontology:
	
	`(not-understood`
	`  :sender i`
	`  :receiver j`
	`  :content ((query-if :sender j :receiver i ...)`
	`  (unknown (ontology www)))`
	`  :language sl)`

6.5.13 propose

Summary:	The action of submitting a proposal to perform a certain action, given certain preconditions.
Message content:	A tuple containing an action description, representing the action that the sender is proposing to perform, and a proposition representing the preconditions on the performance of the action.
Description:	*Propose* is a general-purpose action to make a proposal or respond to an existing proposal during a negotiation process by proposing to perform a given action subject to certain conditions being true. The actual protocol under which the negotiation process is being conducted is known either by prior agreement, or is explicitly stated in the *:protocol* parameter of the message. The proposer (the sender of the *propose*) informs the receiver that the proposer will adopt the intention to perform the action once the given precondition is met, and the receiver notifies the proposer of the receiver's intention that the proposer performs the action. A typical use of the condition attached to the proposal is to specify the price of a bid in an auctioning or negotiation protocol.
Summary Formal Model	$\langle i, \text{propose}(j, \langle i, a\rangle, p\rangle \equiv$ $\quad \langle i, \text{inform}(j, I_j \text{Done}(a, p) \Rightarrow$ $\quad\quad I_i \text{Done}(a, p))\rangle$ *Note: this summary is included here for completeness. For full details, see §8.*

Example	Agent j informs i that it will sell 50 boxes of plums for $200:
	`(propose`
	`  :sender j`
	`  :receiver i`
	`  :content ((action j (sell plum 50))(cost 200))`
	`  :ontology fruit-market`
	`  :in-reply-to proposal2`
	`  :language sl`
	`  ...`
	`)`

6.5.14 query-if

Summary:	The action of asking another agent whether or not a given proposition is true.
Message content:	A proposition.
Description:	*Query-if* is the act of asking another agent whether (it believes that) a given proposition is true. The sending agent is requesting the receiver to *inform* it of the truth of the proposition. The agent performing the *query-if* act: • has no knowledge of the truth value of the proposition • believes that the other agent does know the truth of the proposition.
Summary Formal Model	<*i*, query-if(*j*, ϕ) ≡ <*i*, request(*j*, <*j*, inform-if(i, ϕ)>)> *Note: this summary is included here for completeness. For full details, see §8.*

Example	Agent i asks agent j if j is registered with domain server d1:
	`(query-if`
	`:sender i`
	`:receiver j`
	`:content`
	`(registered (server d1) (agent j))`
	`:reply-with r09`
	`...)`
	Agent j replies that it is not:
	`(inform`
	`:sender j`
	`:receiver i`
	`:content (not (registered (server d1) (agent j)))`
	`:in-reply-to r09`
	`)`

6.5.15 query-ref

Summary:	The action of asking another agent for the object referred to by an expression.
Message content:	A definite descriptor
Description:	*Query-ref* is the act of asking another agent to inform the requestor of the object identified by a definite descriptor. The sending agent is requesting the receiver to perform an *inform* act, containing the object that corresponds to the definite descriptor. The agent performing the *query-ref* act: • does not know which object corresponds to the descriptor • believes that the other agent does know which object corresponds to the descriptor.
Summary Formal Model	$\langle i$, query-ref(j, $\iota x\ \delta(x)$)$\equiv$ $\langle i$, request(j, $\langle j$, inform-ref(i, $\iota x\ \delta(x)$)$\rangle$)$\rangle$ *Note: this summary is included here for completeness. For full details, see §8.*

Example	Agent i asks agent j for its available services:
	`(query-ref`
	`  :sender i`
	`  :receiver j`
	`  :content`
	`  (iota ?x (available-services j ?x))`
	`  ...)`
	j replies that it can reserve trains, planes and automobiles:
	`(inform`
	`  :sender j`
	`  :receiver i`
	`  :content`
	`  (= (iota ?x (available-services j ?x))`
	`  ((reserve-ticket train)`
	`  (reserve-ticket plane)`
	`  (reserve automobile))`
	`  )`
	`  ...)`

6.5.16 refuse

Summary:	The action of refusing to perform a given action, and explaining the reason for the refusal.
Message content:	A tuple, consisting of an action expression and a proposition giving the reason for the refusal.
Description:	The refuse act is an abbreviation for denying (strictly speaking, *disconfirm*ing) that an act is possible for the agent to perform, and stating the reason why that is so. The refuse act is performed when the agent cannot meet all of the preconditions for the action to be carried out, both implicit and explicit. For example, the agent may not know something it is being asked for, or another agent requested an action for which it has insufficient privilege. The agent receiving a refuse act is entitled to believe that: • the action has not been done • the action is not feasible (from the point of view of the sender of the refusal) • the (causal) reason for the refusal is represented by the a proposition which is the third term of the tuple, (which may be the constant true). There is no guarantee that the reason is represented in a way that the receiving agent will understand: it could be a textual error message. However, a co-operative agent will attempt to explain the refusal constructively.
Summary Formal Model	$\langle i, \text{refuse}(j, a, \phi)\rangle \equiv$ $\langle i, \text{disconfirm}(j, \text{Feasible}(a))\rangle$; $\langle i, \text{inform}(j, \phi \wedge (\phi \Rightarrow (\neg \text{Done}(a) \wedge \neg \text{I}_j \text{Done}(a))))\rangle$ *Note: this summary is included here for completeness. For full details, see §8.*

Example	Agent j refuses to i reserve a ticket for i, since i there are insufficient funds in i's account:
	`(refuse`
	`:sender j`
	`:receiver i`
	`:content`
	`(`
	`(action j (reserve-ticket LHR, MUC, 27-sept-97))`
	`(insufficient-funds ac12345)`
	`)`
	`:language sl)`

6.5.17 reject-proposal

Summary:	The action of rejecting a proposal to perform some action during a negotiation.
Message content:	A tuple consisting of an action description and a proposition which formed the original proposal being rejected, and a further proposition which denotes the reason for the rejection.
Description:	*Reject-proposal* is a general-purpose rejection to a previously submitted proposal. The agent sending the rejection informs the receiver that it has no intention that the recipient performs the given action under the given preconditions. The additional proposition represents a reason that the proposal was rejected. Since it is in general hard to relate cause to effect, the formal model below only notes that the reason proposition was believed true by the sender at the time of the rejection. Syntactically the reason on the lhs should be treated as a causal explanation for the rejection, even though this is not established by the formal semantics.
Summary Formal Model	$\langle i,$ reject-proposal($j, \langle j, a\rangle, p(e, \langle j, a\rangle), \phi\rangle \equiv$ $\langle i,$ inform($j, \neg I_i$ (Done($\langle j, a\rangle$) $\wedge$ p(e, <j, a>)) $\wedge\phi$)> *Note: this summary is included here for completeness. For full details, see §8.*

Example	Agent i informs j that it rejects an offer from j to sell
	`(reject-proposal`
	`:sender i`
	`:receiver j`
	`:content ((action j (sell plum 50)) (price-too-high 50))`
	`:in-reply-to proposal13`
	`)`

6.5.18 request

Summary:	The sender requests the receiver to perform some action. One important class of uses of the request act is to request the receiver to perform another communicative act.
Message content:	An action description.
Description:	The sender is requesting the receiver to perform some action. The content of the message is a description of the action to be performed, in some language the receiver understands. The action can be any action the receiver is capable of performing: pick up a box, book a plane flight, change a password etc. An important use of the request act is to build composite conversations between agents, where the actions that are the object of the request act are themselves communicative acts such as *inform*.
Summary Formal Model	<*i*, request(*j*, a)> FP: FP(a) [i\j] $\wedge$ B_i Agent(*j*, a) $\wedge\neg B_i$ I_j Done(a) RE: Done(a) *Note: this summary is included here for completeness. For full details, see §8.*

Examples	Agent i requests j to open a file:
	`(request`
	`:sender i`
	`:receiver j`
	`:content "open \"db.txt\" for input"`
	`:language vb)`

6.5.19 request-when

Summary:	The sender wants the receiver to perform some action when some given proposition becomes true.
Message content:	A tuple of an action description and a proposition.
Description:	*Request-when* allows an agent to inform another agent that a certain action should be performed as soon as a given precondition, expressed as a proposition, becomes true. The agent receiving a *request-when* should either refuse to take on the commitment, or should arrange to ensure that the action will be performed when the condition becomes true. This commitment will persist until such time as it is discharged by the condition becoming true, the requesting agent *cancel*s the *request-when*, or the agent decides that it can no longer honour the commitment, in which case it should send a *refuse* message to the originator. No specific commitment is implied by the specification as to how frequently the proposition is re-evaluated, nor what the lag will be between the proposition becoming true and the action being enacted. Agents which require such specific commitments should negotiate their own agreements prior to submitting the *request-when* act.
Summary Formal Model	$\langle i,$ request-when($j, \langle j,$ a$\rangle$, p)$\rangle \equiv$ $\langle i,$ inform($j, I_i (\exists e)$ Enables(e, $B_j p$) $\Rightarrow$ After(e, $\langle j,$ a$\rangle$))$\rangle$ *Note: this summary is included here for completeness. For full details, see §8.*

Examples	Agent i tells agent j to notify it as soon as an alarm occurs.
	
	`(request-when`
	`  :sender i`
	`  :receiver j`
	`  :content (`
	`  (inform :sender j :receiver i`
	`  :content "something alarming!")`
	`  (Done( alarm ))`
	`  )`
	`  ...`
	`)`

6.5.20 request-whenever

Summary:	The sender wants the receiver to perform some action as soon as some proposition becomes true and thereafter each time the proposition becomes true again.
Message content:	A tuple of an action description and a proposition.
Description:	*Request-whenever* allows an agent to inform another agent that a certain action should be performed as soon as a given precondition, expressed as a proposition, becomes true, and that, furthermore, if the proposition should subsequently become false, the action will be repeated as soon as it once more becomes true. *Request-whenever* represents a persistent commitment to re-evaluate the given proposition and take action when its value changes. The originating agent may subsequently remove this commitment by performing the *cancel* action. No specific commitment is implied by the specification as to how frequently the proposition is re-evaluated, nor what the lag will be between the proposition becoming true and the action being enacted. Agents who require such specific commitments should negotiate their own agreements prior to submitting the *request-when* act.
Summary Formal Model	$\langle i$, request-whenever(j, $\langle j$, a$\rangle$, p)$\rangle \equiv$ $\langle i$, inform(j, I_i Done(a, $(\exists e)$ Enabled(e, B_j p)))$\rangle$ *Note: this summary is included here for completeness. For full details, see §8.*

Examples	Agent i tells agent j to notify it whenever the price of widgets rises from less than 50 to more than 50.
	
	`(request-whenever`
	`  :sender i`
	`  :receiver j`
	`  :content ((inform :sender j :receiver i`
	`  :content (price widget))`
	`  (> (price widget) 50))`
	`  ...`
	`)`

6.5.21 subscribe

Summary:	The act of requesting a persistent intention to notify the sender of the value of a reference, and to notify again whenever the object identified by the reference changes.
Message content:	A definite descriptor
Description:	The *subscribe* act is a persistent version of *query-ref*, such that the agent receiving the *subscribe* will *inform* the sender of the value of the reference, and will continue to send further *informs* if the object denoted by the definite description changes. A subscription set up by a *subscribe* act is terminated by a *cancel* act.
Summary Formal Model	Version 1 (Philippe): <i, subscribe(j, ιx δ(x))> ≡ <i, request-whenever(j, <j, inform-ref(i, ιx δ(x)>, (∃y) Bj ((ιx δ(x)) = y))> Version 2 (old): <i, subscribe(j, ιx δ(x))> ≡ <i, inform(j, I_i (∀e) (∀e') (∀y) Feasible(e; e', Done(e', ¬Bj (ιx δ(x))=y) ∧ Bj (ιx δ(x))=y) ⇒ Feasible(e ; e', (∀e1) Feasible(e1) ⇒ (∃ e2) (∃ e3) (e1 = (e2 ; <j, inform(i, (ιx δ(x)) = y)> ; e3)))> **We need a final decision on this – ed.** *Note: this summary is included here for completeness. For full details, see §8.*

Examples	Agent i wishes to be updated on the exchange rate of Francs to Dollars, and makes a subscription agreement with j (an exchange rate server):
	
	`(subscribe`
	`  :sender i`
	`  :receiver j:`
	`  :content (iota ?x (= ?x (xch-rate FFr USD)))`
	`)`

7 Interaction Protocols

Ongoing conversations between agents often fall into typical patterns. In such cases, certain message sequences are expected, and, at any point in the conversation, other messages are expected to follow. These typical patterns of message exchange are called *protocols*. A designer of agent systems has the choice to make the agents sufficiently aware of the meanings of the messages, and the goals, beliefs and other mental attitudes the agent possesses, that the agent's planning process causes such protocols to arise spontaneously from the agents' choices. This, however, places a heavy burden of capability and complexity on the agent implementation, though it is not an uncommon choice in the agent community at large. An alternative, and very pragmatic, view is to pre-specify the protocols, so that a simpler agent implementation can nevertheless engage in meaningful conversation with other agents, simply by carefully following the known protocol.

This section of the specification details a number of such protocols, in order to facilitate the effective inter-operation of simple and complex agents. No claim is made that this is an exhaustive list of useful protocols, nor that they are necessary for any given application. The protocols are given pre-defined names: the requirement for adhering to the specification is:

> **Requirement 8:**
> An ACL compliant agent need not implement any of the standard protocols, nor is it restricted from using other protocol names. However, if one of the standard protocol names is used, the agent must behave consistently with the protocol specification given here.

Note that, by their nature, agents can engage in multiple dialogues, perhaps with different agents, simultaneously. The term *conversation* is used to denote any particular instance of such a dialogue. Thus, the agent may be concurrently engaged in multiple conversations, with different agents, within different protocols. The remarks in this section which refer to the receipt of messages under the control of a given protocol refer only to a particular conversation.

7.1 Specifying when a protocol is in operation

Notionally, two agents intending to use a protocol should first negotiate whether to use a protocol, and, if so, which one. However, providing the mechanism to do this would negate a key purpose of protocols, which is to simplify the agent implementation. The following convention is therefore adopted: placing the name of the protocol that is being used in the *:protocol* parameter of a message is equivalent to (and slightly more efficient than) prepending with an *inform* that i intends that the protocol will be done (i.e., formally, I_i Done(*protocol-name*)).

Once the protocol is finished, which may occur when one of the final states of the protocol is reached, or when the name of the protocol is dropped from the `:protocol` parameter of the message, this implicit intention has been satisfied.

If the agent receiving a message in the context of a protocol which it cannot, or does not wish to, support, it should send back a *refuse* message explaining this.

Example:

```
(request :sender i
         :receiver j
         :content some-act
         :protocol fipa-request
)
```

7.2 Protocol Description Notation

The following notation is used to describe the standard interaction protocols in a convenient manner:

- Boxes with double edges represent communicative actions.
- White boxes represent actions performed by initiator.
- Shaded boxes are performed by the other participant(s) in the protocol.

— *Italicised text* with no box represents a comment.

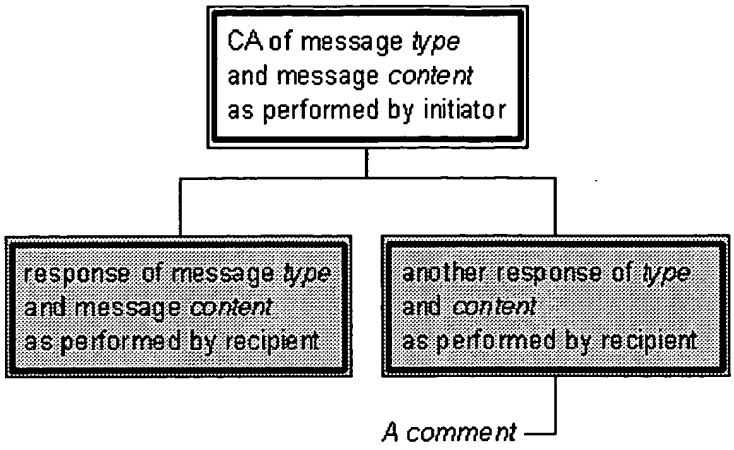


Figure 2 — Example of graphical description of protocols

The above notation is meant solely to represent the protocol as it might be seen by an outside observer. In particular, only those actions should be depicted which are explicit objects of conversation. Actions which are internal to an agent in order to execute the protocol are not represented as this may unduly restrict an agent implementation (e.g. it is of no concern how an agent arrives at a proposal).

7.3 Defined protocols

7.3.1 Failure to understand a response during a protocol

Whilst not, strictly speaking, a protocol, by convention an agent which is expecting a certain set of responses in a protocol, and which receives another message not in that set, should respond with a *not-understood* message.

To guard against the possibility of infinite message loops, it is not permissible to respond to a *not-understood* message with another *not-understood* message!

7.3.2 FIPA-request Protocol

The FIPA-request protocol simply allows one agent to request another to perform some action, and the receiving agent to perform the action or reply, in some way, that it cannot.

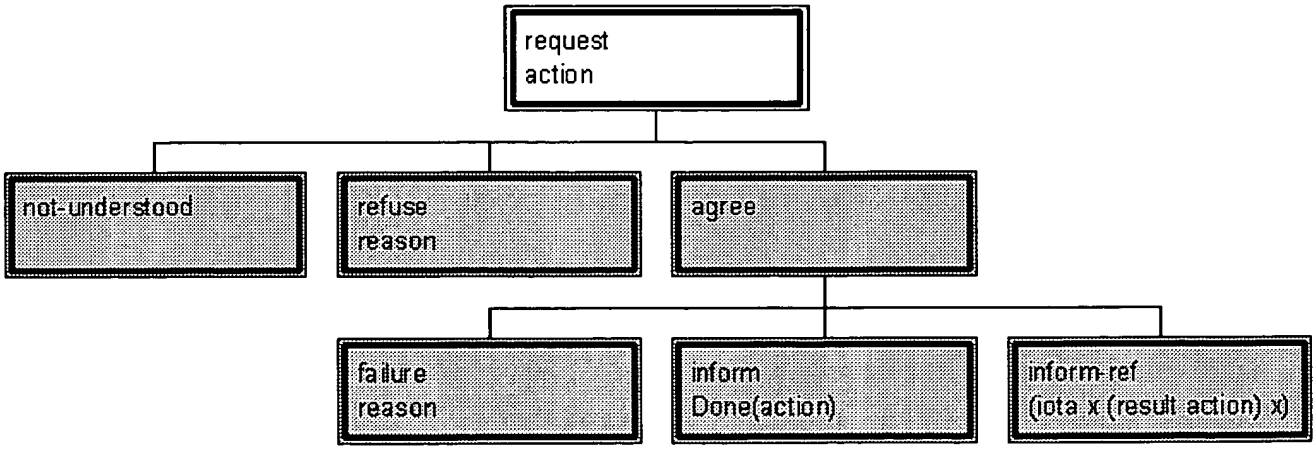


Figure 3 — FIPA-Request Protocol

7.3.3 FIPA-query Protocol

In the FIPA-query protocol, the receiving agent is requested to perform some kind of inform action. Requesting to inform is a query, and there are two query-acts: query-if and query-ref. Either act may be used to initiate this protocol. If the protocol is initiated by a query-if act, it the responder will plan to return the answer to the query with a normal inform act. If initiated by query-ref, it will instead be an inform-ref that is planned. Note that, since inform-ref is a macro act, it will in fact be an inform act that is in fact carried out by the responder.

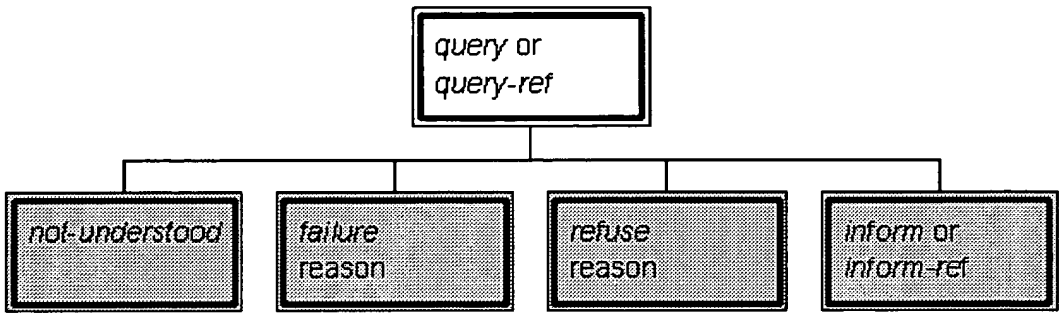


Figure 4 — FIPA-Query Protocol

7.3.4 FIPA-request-when Protocol

The FIPA-request-when protocol is simply an expression of the full intended meaning of the request-when action. The requesting agent uses the *request-when* action to seek from the requested agent that it performs some action in the future once a given precondition becomes true. If the requested agent understands the request and does not refuse, it will wait until the precondition occurs then perform the action, after which it will notify the requester that the action has been performed. Note that this protocol is somewhat redundant in the case that the action requested involves notifying the requesting agent anyway. If it subsequently becomes impossible for the requested agent to perform the action, it will send a refuse request to the original requestor.

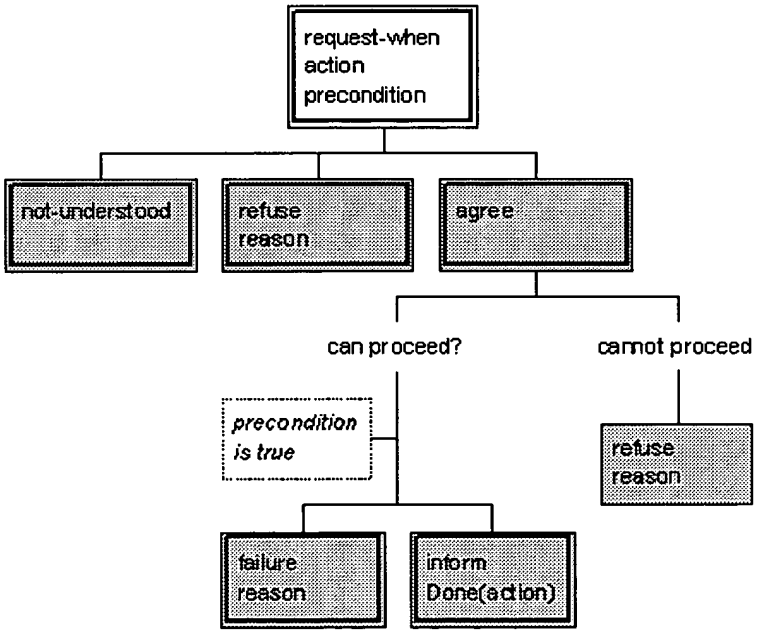


Figure 5 — FIPA-request-when protocol

7.3.5 FIPA-contract-net Protocol

This section presents a version of the widely used *Contract Net Protocol*, originally developed by Smith and Davis [Smith & Davis 80]. FIPA-Contract-Net is a minor modification of the original contract net protocol in that it adds

rejection and *confirmation* communicative acts. In the contract net protocol, one agent takes the role of *manager*. The manager wishes to have some task performed by one or more other agents, and further wishes to optimise a function that characterises the task. This characteristic is commonly expressed as the *price*, in some domain specific way, but could also be soonest time to completion, fair distribution of tasks, etc.

The manager solicits *proposals* from other agents by issuing a *call for proposals*, which specifies the task and any conditions the manager is placing upon the execution of the task. Agents receiving the call for proposals are viewed as potential *contractors*, and are able to generate proposals to perform the task as *propose* acts. The contractor's proposal includes the preconditions that the contractor is setting out for the task, which may be the price, time when the task will be done, etc. Alternatively, the contractor may *refuse* to propose. Once the manager receives back replies from all of the contractors, it evaluates the proposals and makes its choice of which agents will perform the task. One, several, or no agents may be chosen. The agents of the selected proposal(s) will be sent an acceptance message, the others will receive a notice of rejection. The proposals are assumed to be binding on the contractor, so that once the manager accepts the proposal the contractor acquires a commitment to perform the task. Once the contractor has completed the task, it sends a completion message to the manager.

Note that the protocol requires the manager to know when it has received all replies. In the case that a contractor fails to reply with either a *propose* or a *refuse*, the manager may potentially be left waiting indefinitely. To guard against this, the *cfp* includes a deadline by which replies should be received by the manager. Proposals received after the deadline are automatically rejected, with the given reason that the proposal was late.

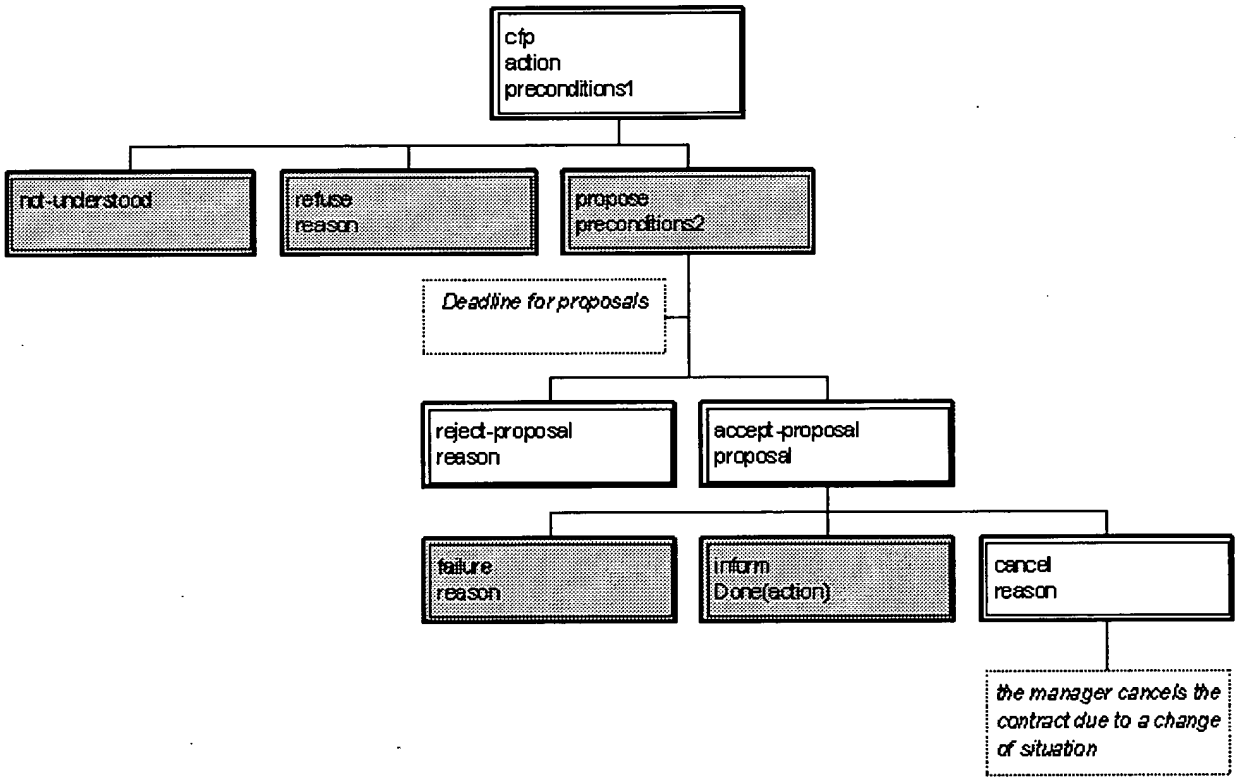


Figure 6 — FIPA-Contract-Net

7.3.6 FIPA-Iterated-Contract-Net Protocol

The *iterated contract net* protocol is an extension of the basic contract net as described above. It differs from the basic version of the contract net by allowing multi-round iterative bidding. As above, the manager issues the initial call for proposals with the *cfp* act. The contractors then answer with their bids as *propose* acts. The manager may then accept one or more of the bids, rejecting the others, or may iterate the process by issuing a revised cfp. The intent is that the manager seeks to get better bids from the contractors by modifying the call and requesting new (equivalently, revised) bids. The process terminates when the manager refuses all proposals and does not issue a new call, accepts one or more of the bids, or the contractors all refuse to bid.

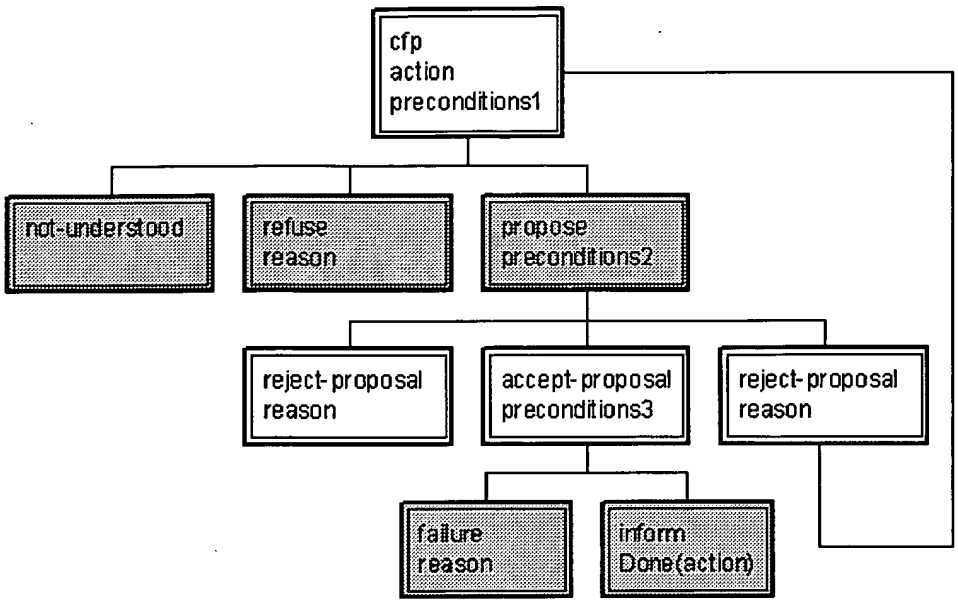


Figure 7 — FIPA-iterated-contract-net protocol

7.3.7 FIPA-Auction-English Protocol

In the English Auction, the auctioneer seeks to find the market price of a good by initially proposing a price below that of the supposed market value, and then gradually raising the price. Each time the price is announced, the auctioneer waits to see if any buyers will signal their willingness to pay the proposed price. As soon as one buyer indicates that it will accept the price, the auctioneer issues a new call for bids with an incremented price. The auction continues until no buyers are prepared to pay the proposed price, at which point the auction ends. If the last price that was accepted by a buyer exceeds the auctioneer's (privately known) reservation price, the good is sold to that buyer for the agreed price. If the last accepted price is less than the reservation price, the good is not sold.

In the following protocol diagram, the auctioneer's calls, expressed as the general *cfp* act, are multicast to all participants in the auction. For simplicity, only one instance of the message is portrayed. Note also that in a physical auction, the presence of the auction participants in one room effectively means that each acceptance of a bid is simultaneously broadcast to all participants, not just the auctioneer. This may not be true in an agent marketplace, in which case it is possible for more than one agent to attempt to bid for the suggested price. Even though the auction will continue for as long as there is at least one bidder, the agents will need to know whether their bid (represented by the *propose* act) has been accepted. Hence the appearance in the protocol of *accept-proposal* and *reject-proposal* messages, despite this being implicit in the English Auction process that is being modelled.

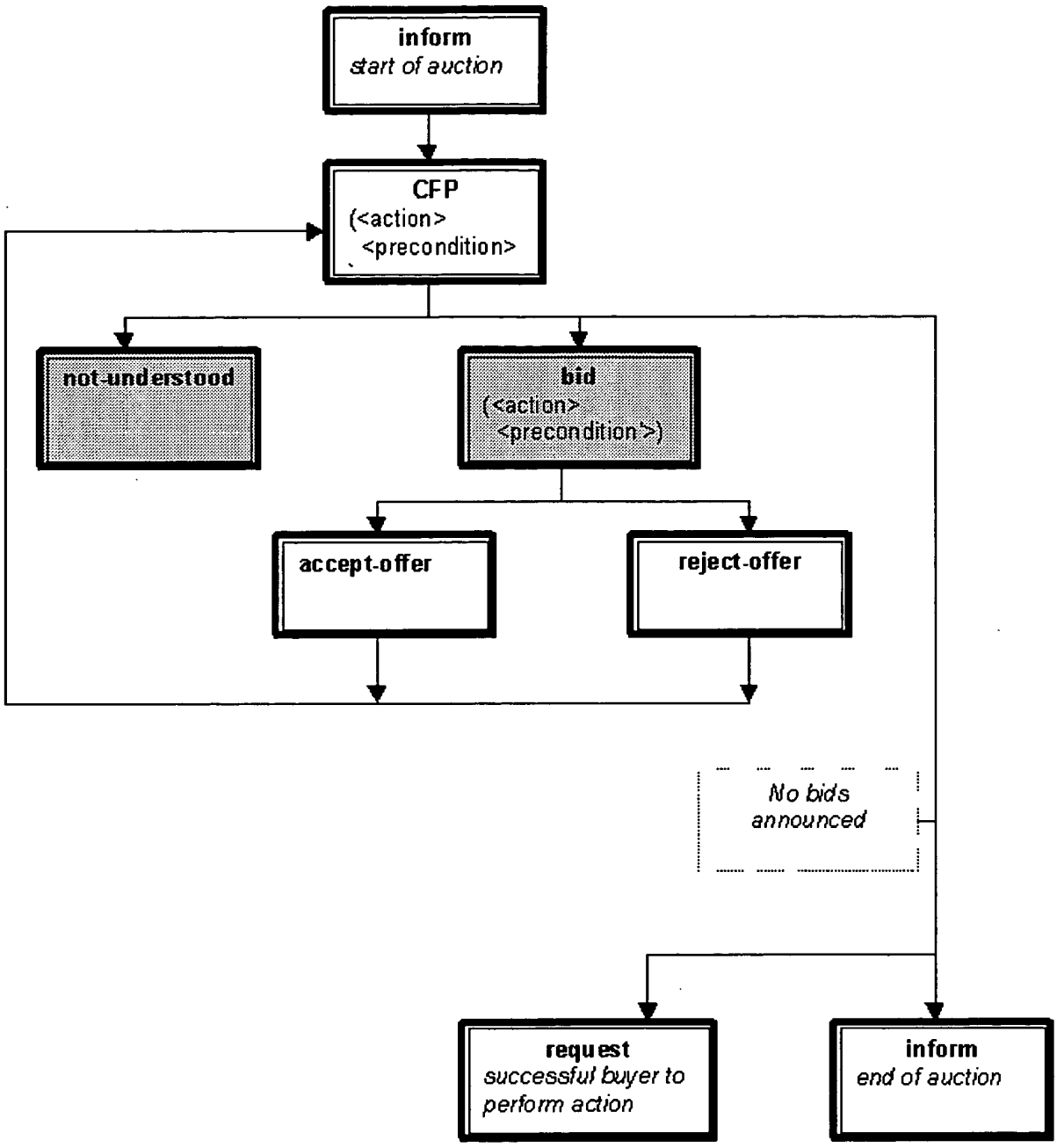


Figure 8 — FIPA-auction-english protocol

7.3.8 FIPA-Auction-Dutch Protocol

In what is commonly called the *Dutch Auction*, the auctioneer attempts to find the market price for a good by starting bidding at a price much higher than the expected market value, then progressively reducing the price until one of the buyers accepts the price. The rate of reduction of the price is up to the auctioneer, and the auctioneer usually has a *reserve price* below which it will not go. If the auction reduces the price to the reserve price with no buyers, the auction terminates.

The term "Dutch Auction" derives from the flower markets in Holland, where this is the dominant means of determining the market value of quantities of (typically) cut flowers. In modelling the actual Dutch flower auction (and indeed in some other markets), some additional complexities occur. First, the good may be split: for example the auctioneer may be selling five boxes of tulips at price *x*, and a buyer may step in and purchase only three of the boxes. The auction then continues, with a price at the next increment below *x*, until the rest of the good is sold or the reserve price met. Such partial sales of goods are only present in some markets; in others the purchaser

must bid to buy the entire good. Secondly, the flower market mechanism is set up to ensure that there is no contention amongst buyers, by preventing any other bids once a single bid has been made for a good. Offers and bids are binding, so there is no protocol for accepting or rejecting a bid. In the agent case, it is not possible to assume, and too restrictive to require, that such conditions apply. Thus it is quite possible that two or more bids are received by the auctioneer for the same good. The protocol below thus allows for a bid to be rejected. This is intended only to be used in the case of multiple, competing, simultaneous bids. It is outside the scope of this specification to pre-specify any particular mechanism for resolving this conflict. In the general case, the agents should make no assumptions beyond "first come, first served". In any given domain, other rules may apply.

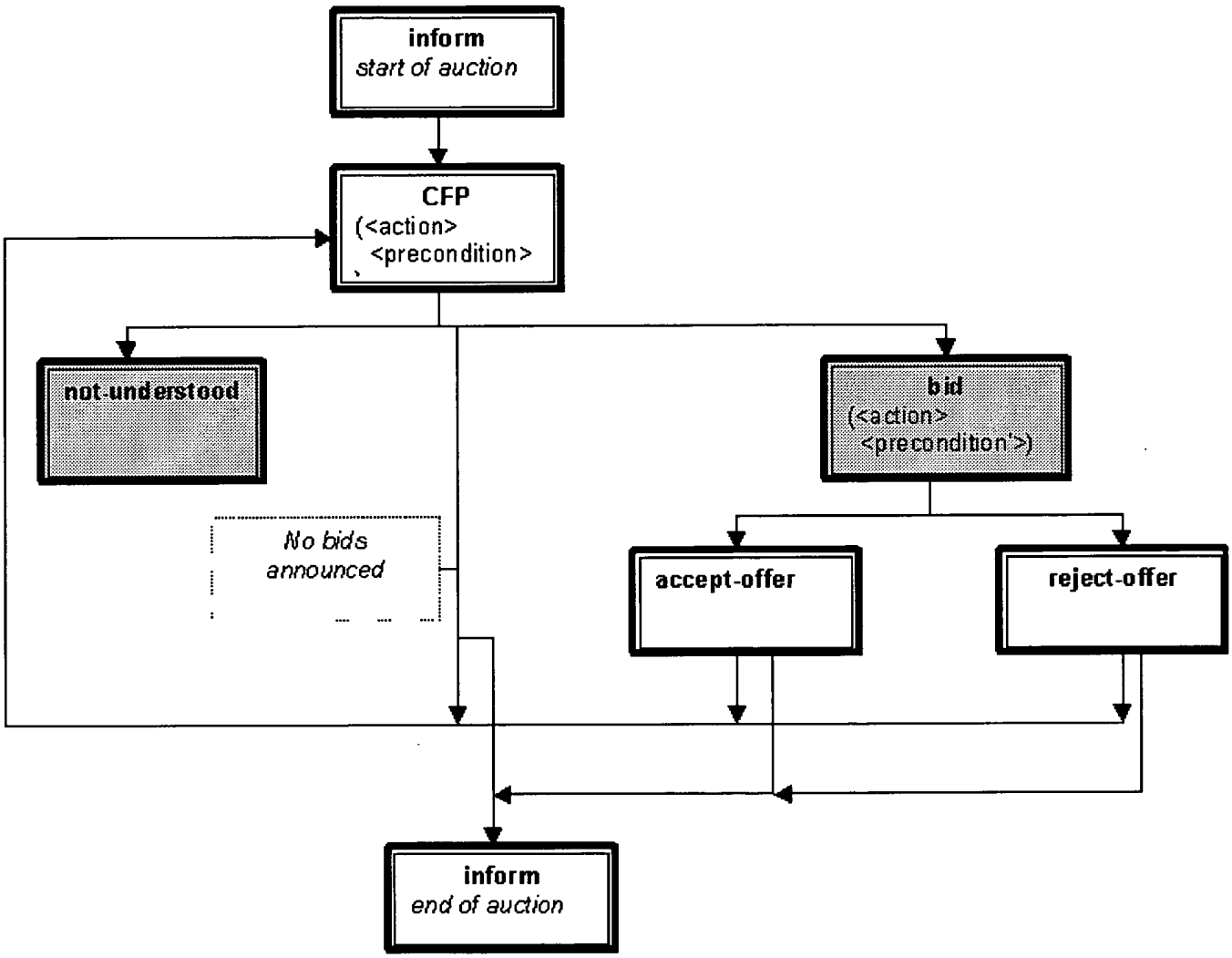


Figure 9 — FIPA-auction-dutch protocol

8 Formal basis of ACL semantics

This section provides a formal definition of the communication language and its semantics. The intention here is to provide a clear, unambiguous reference point for the standardised meaning of the inter-agent communicative acts expressed through messages and protocols. This section of the specification is *normative*, in that agents which claim to conform to the FIPA specification ACL must behave in accordance with the definitions herein. However, this section may be treated as informative in the sense that no new information is introduced here that is not already expressed elsewhere in this document. The non mathematically-inclined reader may safely omit this section without sacrificing a full understanding of the specification.

Note also that *conformance testing*, that is, demonstrating in an unambiguous way that a given agent implementation is correct with respect to this formal model, is not a problem which has been solved in this FIPA specification. Conformance testing will be the subject of further work by FIPA.

8.1 Introduction to formal model

This section presents, in an informal way, the model of communicative acts that underlies the semantics of the message language. This model is presented *only in order to ground the stated meanings* of communicative acts and protocols. It **is not a proposed architecture** or a structural model of the agent design.

Other than the special case of agents that operate singly and interact only with human users or other software interfaces, agents must communicate with each other to perform the tasks for which they are responsible.

Consider the basic case shown below:

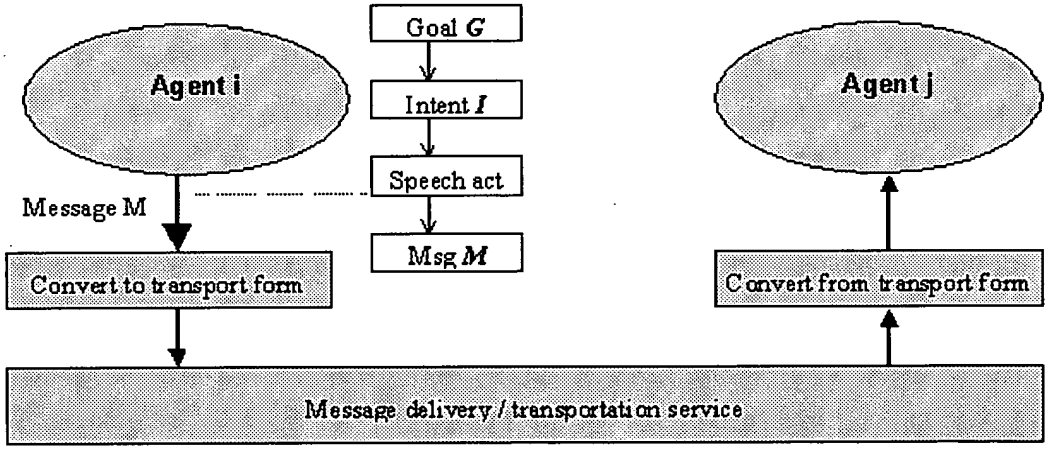


Figure 10 — Message passing between two agents

Suppose that, *in abstract terms*, Agent `i` has amongst its *mental attitudes* the following: some goal or objective `G`, and some intention `I`. Deciding to satisfy `G`, the agent adopts a specific intention, `I`. Note that neither of these statements entail a commitment on the design of `i`: `G` and `I` could equivalently be encoded as explicit terms in the mental structures of a BDI agent, or implicitly in the call stack and programming assumptions of a simple Java or database agent.

Assuming that `i` cannot carry out the intention by itself, the question then becomes which message or set of messages should be sent to another agent (`j` in the figure) to assist or cause intention `I` to be satisfied? If agent `i` is behaving in some reasonable sense rationally, it will not send out a message whose effect will not satisfy the intention and hence achieve the goal. For example, if Harry wishes to have a barbecue (`G = "have a barbecue"`), and thus derives a goal to find out if the weather will be suitable (`G' = "know if it is raining today"`), and thus intends to find out the weather (`I = "find out if it is raining"`), he will be ill-advised to ask Sally "have you bought Acme stock today?". From Harry's perspective, whatever Sally says, it will not help him to determine whether it is raining today.

Continuing the example, if Harry, acting more rationally, asks Sally "can you tell me if it is raining today?", he has acted in a way he hopes will satisfy his intention and meet his goal (assuming that Harry thinks that Sally will know the answer). Harry can reason that the effect of asking Sally is that Sally would tell him, hence making the

request fulfils his intention. Now, having asked the question, can Harry actually assume that, sooner or later, he will know whether it is raining? Harry *can* assume that Sally knows that he does not know, *and* that she knows that he is asking her to tell him. But, simply on the basis of having asked, Harry cannot assume that Sally will act to tell him the weather: she is independent, and may, for example, be busy elsewhere.

In summary: an agent plans, explicitly or implicitly (through the construction of its software) to meet its goals ultimately by communicating with other agents, i.e. sending messages to them and receiving messages from them. The agent will select acts based on the relevance of the act's expected outcome or *rational effect* to its goals. However, *it cannot assume* that the rational effect will necessarily result from sending the messages.

8.2 The SL Language

SL, standing for *Semantic Language*, is the formal language used to define the semantics of the FIPA ACL. As such, SL itself has to be precisely defined. In this section, we present the SL language definition and the semantics of the primitive communicative acts.

8.2.1 Basis of the SL formalism

In SL, logical propositions are expressed in a logic of mental attitudes and actions, formalised in a first order modal language with identity[6] (see [Sadek 91a] for details of this logic). The components of the formalism used in the following are as follows:

- p, p_1, ... are taken to be closed formulas denoting propositions,
- ϕ and ψ are formula schemas, which stand for any closed proposition
- i and j are schematic variables which denote agents
- $\models \phi$ means that ϕ is valid.

The mental model of an agent is based on the representation of three primitive attitudes: *belief*, *uncertainty* and *choice* (or, to some extent, *goal*). They are respectively formalised by the modal operators *B*, *U*, and *C*. Formulas using these operators can be read as:

- $B_i p$ "*i* (implicitly) believes (that) *p*"
- $U_i p$ "*i* is uncertain about *p* but thinks that *p* is more likely than $\neg p$"
- $C_i p$ "*i* desires that *p* currently holds"

The logical model for the operator *B* is a *KD45* possible-worlds-semantics Kripke structure (see, e.g., [Halpern & Moses 85]) with the fixed domain principle (see, *e.g.*, [Garson 84]).

To enable reasoning about action, the universe of discourse involves, in addition to individual objects and agents, sequences of events. A sequence may be formed with a single event. This event may be also the *void* event. The language involves terms (in particular a variable *e*) ranging over the set of event sequences.

To talk about complex *plans*, events (or actions) can be combined to form *action expressions*:

- $a_1 ; a_2$ is a *sequence* in which a_2 follows a_1
- $a_1 \mid a_2$ is a *nondeterministic choice*, in which either a_1 happens or a_2, but not both.

Action expressions will be noted *a*.

The operators *Feasible*, *Done* and *Agent* are introduced to enable reasoning about actions, as follows:

- *Feasible*(*a*, *p*) means that *a* can take place and if it does *p* will be true just after that
- *Done*(*a*, *p*) means that *a* has just taken place and *p* was true just before that
- *Agent*(*i*, *a*) means that *i* denotes the only agent performing, or that will be performing, the actions which appear in action expression *a*.
- *Single(a)* means that *a* denotes an action expression that is not a sequence. Any individual action is *Single*. The composite act `a ; b` is not *Single*. The composite act `a | b` is *Single* iff both a and b are

Single.

From belief, choice and events, the concept of *persistent goal* is defined. An agent *i* has *p* as a persistent goal, if *i* has *p* as a goal and is self-committed toward this goal until *i* comes to believe that the goal is achieved or to believe that it is unachievable. *Intention* is defined as a persistent goal imposing the agent to act. Formulas as $PG_i p$ and $I_i p$ are intended to mean that "*i* has *p* as a persistent goal" and "*i* has the intention to bring about *p*", respectively. The definition of *I* entails that *intention generates a planning process*. See [Sadek 92] for the details of a formal definition of intention.

Note that there is no restriction on the possibility of embedding mental attitude or action operators. For example, formula $U_i B_j I_j Done(a, B_i p)$ informally means that agent *i* believes that, probably, agent *j* thinks that *i* has the intention that action *a* be done before which *i* has to believe *p*.

A fundamental property of the proposed logic is that the modelled agents are perfectly in agreement with their own mental attitudes. Formally, the following schema is valid:

$$\models \phi \Leftrightarrow B_i \phi$$

where ϕ is governed by a modal operator formalising a mental attitude of agent *i*.

8.2.2 Abbreviations

In the text below, the following abbreviations are used:

i) Feasible(a) ≡ Feasible(a, True)

ii) Done(a) ≡ Done(a, True)

iii) Possible(ϕ) ≡ $(\exists a)$Feasible(a, ϕ)

iv) $Bif_i \phi \equiv B_i \phi \vee B_i \neg \phi$

$Bif_i \phi$ means that either agent *i* believes ϕ or that it believes $\neg\phi$.

v) $Bref_i \delta(x) \equiv (\exists y) B_i (\iota x)\delta(x) = y$

where ι is the operator for definite description and $(\iota x)\delta(x)$ is read "the (*x* which is) δ". $Bref_i \delta(x)$ means that agent *i* believes that it knows the (*x* which is) δ.

vi) $Uif_i \phi \equiv U_i \phi \vee U_i \neg \phi$

$Uif_i \phi$ means that either agent *i* is uncertain (in the sense defined above) about ϕ or that it is uncertain about $\neg\phi$.

vii) $Uref_i \delta(x) \equiv (\exists y) U_i (\iota x)\delta(x) = y$

$Uref_i \delta(x)$ has the same meaning as $Bref_i \delta(x)$, except that agent *i* has an uncertainty attitude with respect to $\delta(x)$ instead of a belief attitude

viii) $AB_{n,i,j} \phi \equiv B_i B_j B_i \ldots \phi$

introduces the concept of *alternate beliefs*, *n* is a positive integer representing the number of *B* operators alternating between *i* and *j*.

In the text, the term "knowledge" is used as an abbreviation for "believes or is uncertain of".

8.3 Underlying Semantic Model

The components of a communicative act (CA) model that are involved in a planning process characterise both the reasons for which the act is selected and the conditions that have to be satisfied for the act to be planned. For a given act, the former is referred to as the *rational effect* or *RE*[7], and the latter as the *feasibility preconditions* or *FP*'s, which are the qualifications of the act.

8.3.1 Property 1

To give an agent the capability of planning an act whenever the agent intends to achieve its RE, the agent should adhere to the following property:

Let a_k be an act such that:

i) $(\exists x)\ B_i a_k = x$,

ii) p is the RE of a_k and

iii) $\neg C_i \neg \text{Possible}(\text{Done}(a_k))$;

then the following formula is valid:

$$I_i p \Rightarrow I_i\ \text{Done}(a_1 \mid \ldots \mid a_n)$$

where $a_1, \ldots, a_n$ are *all* the acts of type a_k.

This property says that an agent's intention to achieve a given goal generates an intention that one of the acts known to the agent be done. Further, the act is such that its rational effect corresponds to the agent's goal, and that the agent has no reason for not doing it.

The set of feasibility preconditions for a CA can be split into two subsets: the *ability preconditions* and the *context-relevance preconditions*. The ability preconditions characterise the intrinsic ability of an agent to perform a given CA. For instance, to *sincerely assert* some proposition p, an agent has to believe that p. The context-relevance preconditions characterise the relevance of the act to the context in which it is performed. For instance, an agent can be intrinsically able to make a promise while believing that the promised action is not needed by the addressee. The context-relevance preconditions correspond to the Gricean quantity and relation maxims.

8.3.2 Property 2

This property imposes on an agent an intention to seek the satisfiability of its FP's, whenever the agent elects to perform an act by virtue of property 1 [8]:

$$\models I_i\ \mathit{Done}(a) \Rightarrow B_i\ \mathit{Feasible}(a) \vee I_i B_i\ \mathit{Feasible}(a)$$

8.3.3 Property 3

If an agent has the intention that (the illocutionary component of) a communicative act be performed, it necessarily has the intention to bring about the rational effect of the act. The following property formalises this idea:

$$\models I_i\ \mathit{Done}(a) \Rightarrow I_i\ \mathit{RE}(a)$$

where $\mathit{RE}(a)$ denotes the rational effect of act a.

8.3.4 Property 4

Consider now the complementary aspect of CA planning: the consuming of CA's. When an agent observes a CA, it should believe that the agent performing the act has the intention (to make public its intention) to achieve the rational effect of the act. This is called the *intentional effect*. The following property captures this intuition:

$$\models B_i(\ \mathit{Done}(a) \wedge \mathit{Agent}(j, a) \Rightarrow I_j\ \mathit{RE}(a)\ )$$

Note, for completeness only, that a strictly precise version of this property is as follows:

$$\models B_i(\ \mathit{Done}(a) \wedge \mathit{Agent}(j, a) \Rightarrow I_j\ B_i I_j \mathit{RE}(a)\ )$$

8.3.5 Property 5

Some FP's persist after the corresponding act has been performed. For the particular case of CA's, the next property is valid for all the FP's which do not refer to time. In such cases, when an agent observes a given CA, it is entitled to believe that the persistent feasibility preconditions hold:

$$\models B_i(Done(a) \Rightarrow FP(a))$$

8.4 Notation

A communicative act model will be presented as follows:

<*i*, *Act*(*j*, *C*)>
FP: ϕ_1
RE: ϕ_2

where *i* is the agent of the act, *j* the recipient, *Act* the name of the act, *C* stands for the semantic content or propositional content[9], and ϕ_1 and ϕ_2 are propositions. This notational form is used for brevity, only within this section on the formal basis of ACL. The correspondence to the standard transport syntax adopted above is illustrated by a simple translation of the above example:

```
( Act
    :sender i
    :receiver j
    :content C )
```

Note that this also illustrates that some aspects of the operational use of the FIPA-ACL fall outside the scope of this formal semantics but are still part of the specification. For example, the above example is actually incomplete without `:language` and `:ontology` parameters to given meaning to *C*, or some means of arranging for these to be known.

8.5 Primitive Communicative Acts

8.5.1 The assertive Inform

One of the most interesting assertives regarding the core of mental attitudes it encapsulates is the act of *informing*. An agent *i* is able to *inform* an agent *j* that some proposition *p* is true *only* if *i* believes *p* (*i.e.*, only if $B_i p$). This act is considered to be context-relevant only if *i* does not think that *j* already believes *p* or its negation, or that *j* is uncertain about *p* (recall that belief and uncertainty are mutually exclusive). If *i* is already aware that *j* does already believe *p*, there is no need for further action by *i*. If *i* believes that *j* believes *not p*, *i* should *disconfirm p*. If *j* is uncertain about *p*, *i* should *confirm p*.

<*i*, *INFORM* (*j*, ϕ)>
FP: $B_i\phi \wedge \neg B_i(Bif_j\phi \vee Uif_j\phi)$
RE: $B_j\phi$

The FP's for *inform* have been constructed to ensure mutual exclusiveness between CA's, when more that one CA might deliver the same rational effect.

Note, for completeness only, that the above version of the *Inform* model is the operationalised version. The complete theoretical version (regarding the FP's) is the following:

<*i*, *INFORM* (*j*, ϕ)>
FP: $B_i\phi \wedge \hat{\aleph}_1 \neg AB_{n,i,j} \neg B_i\phi \wedge \neg B_i B_j\phi \wedge \hat{\aleph}_2 \neg AB_{n,i,j} B_j\phi$
RE: $B_j\phi$

8.5.2 The directive Request

The following model defines the directive *Request*:

<*i*, *REQUEST* (*j*, *a*)>
FP: $FP(a)\,[\lambda j] \wedge B_i\, Agent(j, a) \wedge B_i \neg PG_j\, Done(a)$

RE: *Done*(*a*)

where:

— *a* is a schematic variable for which any action expression can be substituted;

— *FP*(*a*) denotes the feasibility preconditions of *a;*

— *FP*(*a*) [*λj*] denotes the part of the FP's of *a* which are mental attitudes of *i*.

8.5.3 Confirming an uncertain proposition: Confirm

The rational effect of the act *Confirm* is identical to that of most of the assertives, *i*.e., the addressee comes to believe the semantic content of the act. An agent *i* is able to *confirm* a property *p* to an agent *j only* if *i* believes *p* (*i.e.*, $B_i p$). This is the sincerity condition an assertive act imposes on the agent performing the act. The act *Confirm* is context-relevant *only* if *i* believes that *j* is uncertain about *p* (*i.e.*, $B_i\, U_j\, p$). In addition, the analysis to determine the qualifications required for an agent to be entitled to perform an *Inform* act remains valid for the case of the act *Confirm*. These qualifications are identical to those of an *Inform* act for the part concerning the ability preconditions, but they are different for the part concerning the context relevance preconditions. Indeed, an act *Confirm* is irrelevant if the agent performing it believes that the addressee is not uncertain of the proposition intended to be *confirmed*.

In view of this analysis, the following is the model for the act *Confirm:*

<i, *CONFIRM*(j, ϕ)>
FP: $B_i \phi \wedge B_i U_j \phi$

RE: $B_j \phi$

8.5.4 Contradicting knowledge: Disconfirm

The *Confirm* act has a negative counterpart: the *Disconfirm* act. The characterisation of this act is similar to that of the *Confirm* act and leads to the following model:

<i, *DISCONFIRM*(j, ϕ)>
FP: $B_i \neg\phi \wedge B_i(U_j \phi \vee B_j \phi)$

RE: $B_j \neg\phi$

8.6 Composite Communicative Acts

An important distinction is made between acts that can be carried out directly, and those macro acts which can be planned (which includes requesting another agent to perform the act), but cannot be directly carried out. The distinction centres on whether it is possible to say that an act has been done, formally *Done(Action, p)* (see §8). An act which is composed of primitive communicative actions (inform, request, confirm), or which is composed from primitive messages by substitution or sequencing (via the ";" operator), can be performed directly and can be said afterwards to be done. For example, agent *i* can inform *j* that *p*; *Done(<i, inform(j, p) >)* is then true, and the meaning (i.e. the *rational effect*) of this action can be precisely stated.

However, a large class of other useful acts is defined by composition using the disjunction operator (written "|"). By the meaning of the operator, only one of the disjunctive components of the act will be performed when the act is carried out. A good example of these macro-acts is the *inform-ref* act. *Inform-ref* is a macro act defined formally by:

$$\langle i, INFORM\text{-}REF(j, \iota x \delta(x))\rangle \equiv \langle i, INFORM(j, \iota x \delta(x) = r_1)\rangle \mid \ldots \mid \langle i, INFORM(j, \iota x \delta(x) = r_n)\rangle$$

where n may be infinite. This act may be requested (for example, j may request i to perform it), or i may plan to

perform the act in order to achieve the (rational) effect of j knowing the referent of $\delta(x)$. However, when the act is actually performed, what is sent, and what can be said to be *Done*, is an *inform* act.

Finally an inter-agent plan is a sequence of such communicative acts, using either composition operator, involving two or more agents. Communications protocols (q.v.) are primary examples of pre-enumerated inter-agent plans.

8.6.1 The closed-question case

In terms of illocutionary acts, exactly what an agent *i* is *requesting* when uttering a sentence such as "Is *p*?" toward a recipient *j*, is that *j* performs the act of "*informing i that p*" or that *j* performs the act "*informing i that* $\neg p$". We know the model for both of these acts: $<j, INFORM\ (i, \phi)>$. In addition, we know the relation "*or*" set between these two acts: it is the relation that allows for the building of action expressions which represent a *non-deterministic choice* between several (sequences of) events or actions.

In fact, as mentioned above, the semantic content of a directive refers to an *action expression*; so, this can be a *disjunction* between two or more acts. Hence, by using the utterance "Is *p*?", what an agent *i requests* an agent *j* to do is the following action expression:

$<j, INFORM\ (i, p)> \mid <j, INFORM\ (i, \neg p)>$

It seems clear that the semantic content of a directive realised by a yes/no-question can be viewed as an action expression characterising an indefinite choice between two CA's *Inform*. In fact, it can also be shown that the binary character of this relation is only a special case: in general, any number of CA's *Inform* can be handled. In this case, the addressee of a directive is allowed to choose one among several acts. This is not only a theoretical generalisation: it accounts for classical linguistic behaviour traditionally called *Alternatives question*. An example of an utterance realising an alternative question is "Would you like to travel in first class, in business class, or in economy class?". In this case, the semantic content of the *request* realised by this utterance is the following action expression:

$<j, INFORM\ (i, p_1)> \mid <j, INFORM\ (i, p_2)> \mid <j, INFORM\ (i, p_3)>$

where p_1, p_2 and p_3 are intended to mean respectively that *j* wants to travel in first class, in business class, or in economy class.

As it stands, the agent designer has to provide the plan-oriented model for this type of action expression. In fact, it would be interesting to have a model which is not specific to the action expressions characterising the non-deterministic choice between CA's of type *Inform*, but a more general model where the actions referred to in the disjunctive relation remain unspecified. In other words, to describe the preconditions and effects of the expression $a_1 \mid a_2 \mid \ldots \mid a_n$ where a_1, a_2, ..., a_n are any action expressions. It is worth mentioning that the goal is to characterise this action expression as a *disjunctive macro-act* which is planned as such; we are not attempting to characterise the non-deterministic choice between acts which are planned separately. In both cases, the result is a branching plan but in the first case, the plan is branching in an *a priori* way while in the second case it is branching in an *a posteriori* way.

An agent will plan a macro-act of non-deterministic choice when it intends to achieve the rational effect of one of the acts composing the choice, *no matter which one it is*. To do that, one of the feasibility preconditions of the acts must be satisfied, *no matter which one it is*. This produces the following model for a disjunctive macro-act:

$a_1 \mid a_2 \mid \ldots \mid a_n$

FP: $FP(a_1) \vee FP(a_2) \vee \ldots \vee FP(a_n)$

RE: $RE(a_1) \vee RE(a_2) \vee \ldots \vee RE(a_n)$

where $FP(a_k)$ and $RE(a_k)$ represent the FP's and the RE of the action expression a_k, respectively.

Because the yes/no-question, as shown, is a particular case of alternatives question, the above model can be specialised to the case of two acts *Inform* having opposite semantic contents. Thus, we get the following model:

$<i, INFORM(j, \phi)> \mid <i, INFORM(j, \neg\phi)>$

FP: $Bif_i\phi \wedge \neg B_i(Bif_j\phi \vee Uif_j\phi)$

RE: $Bif_j \phi$

In the same way, we can derive the disjunctive macro-act model which gathers the acts *Confirm* and *Disconfirm*. We will use the abbreviation <*i*, *CONFDISCONF*(*j*, ϕ)> to refer to the following model:

<*i*, *CONFIRM*(*j*, ϕ)> | <*i*, *DISCONFIRM*(*j*, ϕ)>
FP: $Bif_i \phi \wedge B_i U_i \phi$
RE: $Bif_j \phi$

8.6.2 The query-if act:

Starting from the act models <*j*, *INFORM-IF*(*i*, ϕ)> and <*i*, *REQUEST*(*j*, *a*)>, it is possible to derive the query-if act model (and not plan, as shown below). Unlike a confirm/disconfirm-question, which will be addressed below, an query-if act requires the agent performing it not to have any knowledge about the proposition whose truth value is asked for. To get this model, a transformation[10] has to be applied to the FP's of the act <j, INFORM-IF (i, ϕ)> and leads to the following model for a query-if act:

<*i*, *QUERY-IF*(*j*, ϕ)> ≡ <*i*, *REQUEST*(*j*, <*j*, *INFORM-IF*(*i*, ϕ)>)>
FP: $\neg Bif_i \phi \wedge \neg Uif_i \phi \wedge B_i \neg PG_j$ *Done*(<*j*, *INFORM-IF* (*i*, ϕ)>)
RE: *Done*(<*j*, *INFORM*(*i*, ϕ)> / <*j*, *INFORM*(*i*, $\neg\phi$)>)

8.6.3 The confirm/disconfirm-question act:

In the same way, it is possible to derive the following *Confirm/Disconfirm-question* act model:

<*i*, *REQUEST*(*j*, <*j*, *CONFDISCONF*(*i*, ϕ)>)>
FP: $U_i \phi \wedge B_i \neg PG_j$ *Done*(<*j*, *CONFDISCONF*(*i*, ϕ)>)
RE: *Done*(<*j*, *CONFIRM*(*i*, ϕ)> | <*j*, *DISCONFIRM*(*i*, ϕ)>)

8.6.4 The open-question case:

Open question is a question which does not suggest a choice and, in particular, which does not require a yes/no answer. A particular case of open questions are the questions which require referring expressions as an answer. They are generally called *wh-questions*. The "wh" refers to interrogative pronouns such as "what", "who", "where", or "when". Nevertheless, this must not be taken literally since the utterance "How did you travel?" can be considered as a wh-question.

A formal plan-oriented model for the wh-questions is required. In the model below, *from the addressee's viewpoint*, this type of question can be viewed as a closed question where the suggested choice is not made explicit because it is *too wide*. Indeed, a question such as "What is your destination?" can be restated as "What is your destination: Paris, Rome,... ?".

The problem is that, in general, the set of definite descriptions among which the addressee can (and must) choose is potentially an infinite set, not because, referring to the example above, there may be an infinite number of destinations, but because, theoretically, each destination can be referred to in potentially an infinite number of ways. For instance, Paris can be referred to as "the capital of France", "the city where the Eiffel Tower is located", "the capital of the country where the Man-Rights Chart was founded", *etc.* However, it must be noted that in the context of man-machine communication, the language used is finite and hence the number of descriptions acceptable as an answer to a *wh-question* is also finite.

When asking a *wh-question*, an agent *j* intends to acquire from the addressee *i* an identifying referring expression (IRE) [Sadek 90] for a definite description, in the general case. Therefore, agent *j* intends to make his interlocutor *i* perform a CA which is of the following form:

<*i*, *INFORM*(*j*, $\iota x \delta(x) = r$)>

where *r* is an IRE (*e.g.*, a standard name or a definite description) and $\iota x \delta(x)$ is a definite description. Thus, the semantic content of the directive performed by a wh-question is a disjunctive macro-act composed with acts of the form of the act above. Here is the model of such a macro-act:

$<i, INFORM(\ j, \iota x \delta(x) = r_1\ )> \mid ... \mid <i, INFORM(\ j, \iota x \delta(x) = r_k\ )>$

where r_k are IREs. To deal with the case of closed questions, the generic plan-oriented model proposed for a disjunctive macro-act can be instantiated for the account of the macro-act above. Note that the following equivalence is valid:

$$(B_i\ \iota x \delta(x) = r_1 \vee B_i\ \iota x \delta(x) = r_2 \vee ...\ ) \Leftrightarrow (\exists y)\ B_i\ \iota x \delta(x) = y$$

This produces the following model, which is referred to as $<i, INFORM\text{-}REF(\ j, \iota x\ \delta(x)\ )>$:

$<j, INFORM\text{-}REF(\ i, \iota x\ \delta(x)\ )>$

FP: $Bref_i\ \delta(x) \wedge \neg B_i \alpha ref_j\ \delta(x)$

RE: $Bref_j\ \delta(x)$

where $Bref_j\ \delta(x)$ and $Uref_j\ \delta(x)$ are abbreviations introduced above, and $\alpha ref_j \delta(x)$ is an abbreviation defined as:

$$\alpha ref_j\ \delta(x) \equiv Bref_j\ \delta(x) \vee Uref_j\ \delta(x)$$

Provided the act models $<j, INFORM\text{-}REF\ (i, \iota x\ \delta(x))>$ and $<i, REQUEST\ (j, a)>$, *the wh-question* act model can be built up in the same way as for the yn-question act model. Applying the same transformation to the FP's of the act schema $<j, INFORM\text{-}REF\ (i, \iota x \delta(x))>$, and by virtue of property 3, the following model is derived:

$<i, REQUEST(\ j, <j, INFORM\text{-}REF(\ i, \iota x\ \delta(x)>\ )>$

FP: $\neg \alpha ref_i\ \delta(x) \wedge B_i \neg PG_j\ Done(\ <j, INFORM\text{-}REF(\ i, \iota x \delta(x)\ )>)$

RE: $Done(\ <i, INFORM\ (j, \iota x \delta(x) = r_1\ )> \mid ... \mid <i, INFORM(\ j, \iota x \delta(x) = r_k\ )>\ )$

8.6.5 Summary definitions for all standard communicative acts

8.6.5.1 Supporting definitions

$Enables(\ e, p\ ) \equiv$
$\quad Done(\ e, \neg p\ ) \wedge p$

$After(\ e_1, e_2\ ) \equiv$
$\quad Done(\ e_1\ ) \wedge$
$\quad (\forall e')\ Feasible(\ e', (\exists f)\ (f = e_1\ ;\ e_2\ ;\ e') \vee (f = e'\ ;\ e_2)\ )$

$Before(\ e_1, e_2\ ) \equiv$
$\quad After(\ e_2, e_1\ )$

$Will\text{-}occur\text{-}when(\ x, p(\ e, x\ )\ ) \equiv$
$\quad (\exists e')\ Done(\ e'\ ;\ x, Feasible(\ e', p(\ e', x\ )\ )\ )$

$Enabled(\ e, p\ ) \equiv$
$\quad Done(\ e, \neg p\ ) \wedge p$

8.6.5.2 Agree

To be completed.

8.6.5.3 Accept-proposal

$<i,$ accept-proposal$(\ j, <j, a>, p(\ e, <j, a>\ )\ )> \equiv$
$\quad <i,$ inform$(\ j, I_i$ Will-occur-when$(\ <j, a>, p(\ e, <j, a>\ )\ )\ )>$

i informs j that i has the intention that j will perform action a just as soon as the precondition parameterised by the

action and some event in the future becomes true.

8.6.5.4 Propose

$\langle i, \text{propose}(j, \langle i, a\rangle, p(e, \langle i, a\rangle)\rangle \equiv$
$\quad \langle i, \text{inform}(j, \forall e \text{ Feasible}(e, \text{Done}(\langle j, \text{inform}(i, I_j \text{ Done}(\langle i, a\rangle) \wedge$
$\quad\quad p(e, \langle i, a\rangle) \Rightarrow I_i \text{ Done}(\langle i, a\rangle) \wedge$
$\quad\quad\quad \text{Feasible}(\langle i, a\rangle))) \rangle$

i informs j that, once j informs i that j has adopted the intention for i to perform action a, and the preconditions for i performing a have been established, it will be feasible for i to perform a and i will adopt the intention to perform a.

8.6.5.5 Cancel

$\langle i, \text{cancel}(j, a)\rangle \equiv$
$\quad \langle i, \text{disconfirm}(j, I_i \text{ Done}(a)\rangle$

Cancel is the action of cancelling any form of *requested* action. In other words, an agent *i* has requested an agent *j* to perform some action, possibly if some condition holds. This has the effect of *i* informing *j* that *i* has an intention. When *i* comes to drop its intention, it has to inform *j* that it no longer has this intention, i.e. a *disconfirm*.

8.6.5.6 cfp

$\langle i, \text{cfp}(j, \langle j, a\rangle, p(e, \langle j, a\rangle))\rangle \equiv$
$\quad \langle i, \text{query-ref}(j, \iota x$
$\quad\quad (I_j \forall e \text{ Feasible}(e, \text{Done}(e ; \langle i, \text{inform}(j, I_i \langle j, a\rangle) \rangle) \wedge$
$\quad\quad\quad ((x = p'(e, \langle j, a\rangle)) \wedge p(e, \langle j, a\rangle)$
$\quad\quad\quad\quad \Rightarrow$
$\quad\quad\quad I_j \text{ Done}(\langle j, a\rangle) \wedge \text{Feasible}(\langle j, a\rangle)))) \rangle$

i requests j to inform i of the additional preconditions (i.e. predicate p') j would require before performing the action a with i's preconditions (i.e. predicate p).

8.6.5.7 confirm

$\langle i, \text{confirm}(j, \phi)\rangle$
$\quad$ FP: $B_i\phi \wedge B_i U_j\phi$
$\quad$ RE: $B_j\phi$

Confirm is a primitive communicative act.

8.6.5.8 Disconfirm

$\langle i, \text{disconfirm}(j, \phi)\rangle$
$\quad$ FP: $B_i\neg\phi \wedge B_i(U_j\phi \vee B_j\phi)$
$\quad$ RE: $B_j\neg\phi$

Disconfirm is a primitive communicative act.

8.6.5.9 Failure

$\langle i, \text{failure}(j, a, p)\rangle \equiv$
$\quad \langle i, \text{inform}(j, (\exists e) \text{Single}(e) \wedge \text{Done}(e, I_i \text{ Done}(a)) \wedge p \wedge$
$\quad\quad (p \Rightarrow (\neg\text{Done}(a) \wedge \neg I_i \text{ Done}(a))))\rangle$

i informs j that, in the past, i had the intention to do action a, but because p was true, a was not done and i no longer has the intention to do a.

8.6.5.10 inform

$\langle i, \text{inform}(j, \phi)\rangle$
$\quad$ FP: $B_i\phi \wedge \neg B_i(\text{Bif}_j\phi \vee \text{Uif}_j\phi)$

RE: $B_j\phi$

Inform is a primitive communicative act.

8.6.5.11 inform-if

<*i*, inform-if(*j*, p)> ≡
<*i*, inform(*j*, p)> | <*i*, inform(*j*, ¬p)>

Inform-if represents two possible courses of action: i informs j that p, or i informs j that not p.

8.6.5.12 inform-ref

<*i*, inform-ref(*j*, ιx δ(x))> ≡
<*i*, inform(*j*, ιx δ(x) = r_1)> | ... | <*i*, inform(*j*, ιx δ(x) = r_k)>

Inform-ref represents an unbounded, possibly infinite set of possible courses of action, in which i informs j of the referent of x.

8.6.5.13 query-if

<*i*, query-if(*j*, ϕ) ≡
<*i*, request(*j*, <*j*, inform-if(i, ϕ)>)>

i requests j that j informs i whether or not ϕ is true.

8.6.5.14 query-ref

<*i*, query-ref(*j*, ιx δ(x))≡
<*i*, request(*j*, <*j*, inform-ref(*i*, ιx δ(x))>)>

i requests j that j informs i of the referent of x

8.6.5.15 refuse

<*i*, refuse(*j*, a, ϕ)> ≡
<*i*, disconfirm(*j*, Feasible(a))> ;
<*i*, inform(*j*, ϕ∧ (ϕ⇒ (¬Done(a) ∧¬I_i Done(a))))>

i informs j that action a is not feasible, and further that, because of proposition ϕ, a has not been done and i has no intention to do a.

8.6.5.16 reject-proposal

<*i*, reject(*j*, <*j*, a>, ϕ)> ≡
<*i*, inform(*j*, ϕ∧ϕ⇒¬I_i Done(< *j*, a>))>

i informs j that, because of proposition ϕ, i does not have the intention for j to perform action a.

8.6.5.17 request

<*i*, request(*j*, a)>
FP: FP(a) [i\j] ∧ B_i Agent(*j*, a) ∧¬B_i PG_j Done(a)
RE: Done(a)

Request is a primitive communicative act.

8.6.5.18 request-when

<*i*, request-when(*j*, <*j*, a>, p)> ≡
<*i*, inform(*j*, I_i (∃e) Enables(e, B_jp) ⇒ After(e, <*j*, a>))>

i informs j that i intends that, when some event happens that enables j to believe p, that event will be followed by j performing action a.

8.6.5.19 Request-whenever

<*i*, request-whenever(*j*, <*j*, a>, p)> ≡
<*i*, inform(*j*, I_i Done(a, (∃e) Enables(e, B_j p)))>

i informs *j* that *i* intends that *j* will not believe *p* until *j* comes to believe *p* and also performs *a*.

8.6.5.20 subscribe

$\langle i, \text{subscribe}(\, j, \iota x\, \delta(x)\,)\rangle \equiv$
$\langle i, \text{inform}(j, I_i\, (\forall e)\, (\forall e')\, (\forall y)$
Feasible(e; e',
Done(e', $\neg Bj\ (i\ x\ \delta(x))=y) \wedge$
$Bj\ (ix\ \delta(x))=y) \Rightarrow$
Feasible(e ; e', $(\forall e1)$ Feasible(e1) $\Rightarrow$
$(\exists\ e2)\ (\exists\ e3)$
(e1 = (e2 ; <j, inform(i, $(ix\ \delta(x)) = y)$> ; e3))))>

8.6.5.21 not-understood

$\langle i, \text{not-understood}(\, j, a\,)\rangle \equiv$
FP: to be completed
RE: to be completed

not-understood is a primitive communicative act.

8.7 Inter-agent Communication Plans

The properties of rational behaviour stated above in the definitions of the concepts of rational effect and of feasibility preconditions for CA'S suggest an algorithm for CA planning. A plan is built up by this algorithm builds up through the inference of causal chain of intentions, resulting from the application of properties 1 and 2.

With this method, it can be shown that what are usually called "*dialogue acts*" and for which models are postulated, are, in fact, complex plans of interaction. These plans can be derived from primitive acts, by using the principles of rational behaviour. The following is an example of how such plans are derived.

The interaction plan "hidden" behind a question act can be more or less complex depending on the agent mental state when the plan is generated.

Let a *direct question* be a question underlain by a plan which is limited to the reaction strictly legitimised by the question. Suppose that the main content of *i*'s mental state is:

$$B_I\ Bif_j\ \phi,$$
$$I_i\ Bif_i\ \phi$$

By virtue of property 1, the intention is generated that the act $\langle j, \textit{INFORM-IF}(\, i, \phi\,)\rangle$ be performed. Then, according to property 2, there follows the intention to bring about the feasibility of this act. Then, the problem is to know whether the following belief can be derived at that time from *i*'s mental state:

$$B_i(\ Bif_j\ \phi \wedge (\neg B_j\ Bif_i \phi \vee Uif_i\ \phi))$$

This is the case with *i*'s mental state. By virtue of properties 1 and 2, the intention that the act $\langle i, \textit{REQUEST}\,(\, j, \langle j, \textit{INFORM-IF}\,(i, \phi)\rangle\,)\rangle$ be done and then the intention to achieve its feasibility, are inferred. The following belief is derivable:

$$B_i(\ \neg Bif_i\ \phi \wedge \neg Uif_i\ \phi)$$

Now, no intention can be inferred. This terminates the planning process. The performance of a direct strict-yn-question plan can be started by uttering a sentence such as "Has the flight from Paris arrived?", for example.

Given the FP's and the RE of the plan above, the following model for a *direct strict-yn-question plan* can be established:

$\langle i, \textit{YNQUESTION}(\, j, \phi\,)\rangle$
FP: $B_i\ Bif_j\ \phi \wedge \neg Bif_i\ \phi \wedge \neg Uif_i\ \phi \wedge B_i\ \neg B_j(\ Bif_i\ \phi \vee Uif_i\ \phi\,)$

RE: $Bif_{i}\phi$

9 References

To be completed.

[Austin 62] Austin J. L. How to Do Things with Words. *Clarendon Press* 1962

[Cohen & Levesque 90] Cohen P.R. & Levesque H.J. Intention is choice with commitment. *Artificial Intelligence,* 42(2-3):213--262, 1990.

[Cohen & Levesque 95] Cohen P.R. & Levesque H.J. Communicative actions for artificial agents; *Proceedings of the First International Conference on Multi-agent Systems (ICMAS'95)*, San Francisco, CA, 1995.

[Finin et al 97] Finin T., Labrou Y. & Mayfield J., KQML as an **agent communication language**, Bradshaw J. ed., *Sofware agents*, MIT Press, Cambridge, 1997.

[Freed & Borenstein 1996] Freed N & Borenstein N. Multipurpose Internet Mail Extensions (MIME) Part One: Format of the Internet Message Bodies. Internic RFC2045. `ftp://ds.internic.net/rfc/rfc2045.txt`

[Genesereth & Fikes 92] Genesereth M.R. & Fikes R.E. Knowledge interchange format. *Technical report Logic-92-1*, CS Department, Stanford University, 1992.

[Garson 84] Garson, G.W. Quantification in modal logic. In Gabbay, D., & Guentner, F. eds. *Handbook of philosophical logic, Volume II: Extensions of classical Logic.* D. Reidel Publishing Company: 249-307. 1984.

[Guinchiglia & Sebastiani 97] Guinchiglia F. & Sebastiani R., Building decision procedures for modal logics from propositional decision procedures: a case study of Modal K. *Proceedings of CADE 13*, published in Lecture Notes in Artificial Intelligence. 1997.

[Halpern & Moses 85] Halpern, J.Y., & Moses Y. A guide to the modal logics of knowledge and belief: a preliminary draft. *Proceedings of the* IJCAI-85, Los Angeles, CA. 1985.

[KQML93] External Interfaces Working Group, Specification of the KQML **agent-communication language**, 1993.

[Labrou & Finin 94] Labrou Y. & Finin T., A semantic approach for KQML - A general purpose communication language for software agents, *Proceedings of the 3rd International Conference on Information Knowledge Management*, November 1994.

[Labrou 96] Labrou Y. Semantics for an **agent communication language**. *PhD thesis dissertation submission,* University of Maryland Graduate School, Baltimore, September, 1996.

[Sadek 90] Sadek M.D., Logical task modelling for Man-machine dialogue. *Proceedings of AAAI'90*: 970-975, Boston, MA, 1990.

[Sadek 91a] Sadek M.D. Attitudes mentales et interaction rationnelle: vers une théorie formelle de la communication. *Thèse de Doctorat Informatique, Université de Rennes I,* France, 1991.

[Sadek 91b] Sadek M.D. Dialogue acts are rational plans. *Proceedings of the ESCA/ETRW Workshop on the structure of multimodal dialogue,* pages 1-29, Maratea, Italy, 1991.

[Sadek 92] Sadek M.D. A study in the logic of intention. *Proceedings of the 3rd Conference on Principles of Knowledge Representation and Reasoning (KR'92),* pages 462-473, Cambridge, MA, 1992.

[Sadek et al 95] Sadek M.D., Bretier P., Cadoret V., Cozannet A., Dupont P., Ferrieux A., & Panaget F. A co-operative spoken dialogue system based on a rational agent model: A first implementation on the AGS application. *Proceedings of the ESCA/ETR Workshop on Spoken Dialogue Systems : Theories and Applications*, Vigso, Denmark, 1995.

[Searle 69] Searle J.R. *Speech Acts*, Cambridge University Press, 1969.

Additional suggested reading

[Bretier & Sadek 96] Bretier P. & Sadek D. A rational agent as the kernel of a cooperative spoken dialogue system: Implementing a logical theory of interaction. In Muller J.P., Wooldridge M.J., and Jennings N.R. (eds) *Intelligent agents III - Proceedings of the third ATAL*, LNAI, 1996.

[Sadek 94] Sadek M. D. Belief reconstruction in communication. *Speech Communication Journal'94, special issue on Spoken Dialogue*, 15(3-4), 1994.

[Sadek et al 97] Sadek M. D., P. Bretier, & F. Panaget. ARTIMIS: Natural language meets rational agency. *Proceedings of IJCAI '97*, Nagoya, Japan, 1997.

Annex A
(informative)

ACL Conventions and Examples

This annex describes certain conventions that, while not a mandatory part of the specification, are commonly adopted practices that aid effective inter-agent communications. This annex will also serve to provide examples of ACL usage for illustrative purposes.

A.1 Conventions

A.1.1 Conversations amongst multiple parties in agent communities

There is commonly a need in inter-agent dialogues to involve more than two parties in the conversation. A typical example would be of agent *i* posing a question to agent *j* by sending a *query-if* message. Agent *i* believes that *j* is able to answer the query, but in fact *j* finds it necessary to delegate some or all of the task of answering the question to another agent *k*.

The formal definition of the query-if communicative act reads that *i* is requesting *j* that *j informs i* of the truth of proposition *p*. Therefore, even if *j* does delegate all of the query to *k*, the semantics of ACL requires that *j* will be the one to perform the act of informing *i*. *K* cannot inform *i* directly. By extension, any chain of such delegation acts will have to be unwound in reverse order to conform to the current specification.

The restriction that a delegating agent in such a scenario must, in effect, remain "in the loop" clearly does not alter the meaning of the act (except, perhaps, that it exposes *i* to the existence of *k*), but it can be critiqued on the grounds of overall efficiency. A future version of this specification may generalise the semantic definition to allow delegation which includes passing responsibility for answering the originator of the request directly.

See also §A.1.4 *Negotiating by exchange of goals*.

A.1.2 Maintaining threads of conversation

Agents are frequently implemented with the ability to participate in more than one conversation at the same time. These conversations may all be with different agents, or may be with the same agent but in the context of different tasks or subjects. The internal representation and maintenance of structures to manage the separate conversations is a matter for the agent designer. However, there must be some support in the ACL for the concept of separate conversations, else an agent will have no standardised way of disambiguating the conversational context in which to interpret a given message. ACL supports conversation threading through the use of standard message parameters which agents are free (but not required) to use. These are: *:reply-with*, *:in-reply-to* and *:conversation*. Additional contextual information to assist the agent to interpret the meaning of a message is provided through the protocol identifier, *:protocol*.

The first case is one of annotating a message which is expected to generate a response with an expression which serves to abbreviate the context of the enquiry. This abbreviation is then cross-referenced in the reply. For example, agent i asks agent j if the summer in England was wet. Without any ability to refer back to the question, j cannot simply say "yes" because that would be potentially ambiguous. J can disambiguate its reply by saying "yes, the summer in England was wet", or it could say "in response to your question, the answer is yes". Different styles and implementations of agents might adopt either of these tactics. The latter case is performed through the use of *:reply-with* and *:in-reply-to*. The *:reply-with* parameter is used to introduce an abbreviation for the query, *:in-reply-to* is used to refer back to it. For example:

```
(ask-if
    :sender I
    :receiver j
    :content (= (weather England (summer 1997)) wet)
    :ontology meteorology
    :reply-with query-17)
```

```
(inform
    :sender j
    :receiver I
    :content true
    :in-reply-to query-17)
```

In addition to maintaining context over instances of exchanges of communicative acts, the agents may also wish to maintain a longer lived conversational structure. They may be exchanging information about the weather in the UK, and at the same time be discussing that of Peru. The conversation can provide additional interpretative context: for example the question "what was the weather like in the summer?" is meaningful in the context of a conversation about UK meteorology, and rather less so if no such context is known. In addition, the conversation may simply be used by the agent to manage its communication activities, particularly if conversations are strongly link to current tasks. The parameter *:conversation-id* is used to denote a word which identifies the conversation.

A.1.3 Initiating sub-conversations within protocols

The use of protocols (c.f. §7) in agent interactions is introduced in order to provide a tool that facilitates the simplification of the design of some agents, since the agent can expect to know which messages are likely to be received or need to be generated at each stage of the conversation. However, this simplicity can also be restrictive: there may legitimately be cause to step outside the prescribed bounds of the protocol. For example, in a contract net protocol, the manager sends out a *cfp* message, which should normally be followed by a *propose* or a refusal. Suppose that the contractor, however, wishes some additional information (perhaps a clarification). Replying to the *cfp* with, for example, a *query-if* action would break the protocol. While agents with powerful, complete reasoning capabilities can be expected to deal appropriately with such an occurrence, simpler agents, adhering closely to the protocol, may not. Nor is it a solution to anticipate all such likely responses in the protocol: such anticipation is unlikely to cover every possibility, and anyway the resulting complexity would defeat the primary purpose of the protocol.

Instead, the convention is suggested that adopting a new conversation-id (see above) for a reply is sufficient to indicate to the receiver that the reply should not be considered the next step in the protocol. It should not cause a not-understood message to be generated (the normal occurrence if a protocol is broken unexpectedly). A problem remains that adopting a new conversation-id does not make available to the agents involved the convenience of knowing that a rich context is shared. This release of the specification does not address the issue of structured conversation-id's, in which the idea of a context-sharing sub-conversation is supported, though a future version may do so. In the interim, it is suggested that, where a given domain finds that this capability is a necessity, a domain specific solution to the problem of defining conversation-id's is adopted.

A.1.4 Negotiating by exchange of goals

A common practice amongst agent communities is to interact and negotiate at the level of goals and commitments, rather than explicit commands. Indeed, some researchers will say that such *indirect manipulation* is one of the most compelling arguments for the effectiveness of the agent technology paradigm.

While the ACL semantics does include a concept of goal and intention, the core communicative act for influencing another agent's behaviour is the *request* action. The main argument to request is an action, not a goal, which requires the requesting agent to be aware of the actions that another agent can perform, and to plan accordingly. In many instances, the agent may wish to communicate its objectives, and leave the reasoning and planning towards the achievement of those objectives to the recipient agent.

Since no *achieve-goal* action is currently built-in to the ACL, it is common to embed the goal in an expression in the chosen content language which expresses the *action of achieving the goal*. This action can then be requested by the sending agent. Precise details of such a goal encoding depend on the chosen content language. An example might be:

```
(request
    :sender i
    :receiver j
    :content (achieve (at (location 12 84) box17))
    :ontology factory-management
    :reply-with query-17)
```

Note, for symmetry, that a converse domain action `achieved` can also be used to map actions to goals.

A.2 Additional examples

A.2.1 Actions and results

In general, the semantic model underlying the ACL states that an action does not have a value. Clearly all actions have effects, which are causally related to the performance of the action. However, it may be difficult or impossible to determine the causal effects of an action. Even *a posterori* observation may not be able to determine all of the effects of an action. Thus, in general, actions do not have a result. SL allows the capture of some intuitive notions about the effects of actions by associating the occurrence of the action with statements about the state of the world through the *Done* and *Feasible* operators.

However, there is a class of actions which are defined as computational activities, in which it is useful to say that the action has a result. For example, the action of adding two and two in a computational device. These actions are related to the result they produce through the result predicate, which is the remit of a content language and given domain theory. In defining the result predicate, it should be noted that it takes as an argument a term, not an action which is a separate category.

Consider the following three example actions:

```
A:  (request    :sender i  :receiver j
        :content (action j action))

B:  (query-ref  :sender i :receiver j
        :content (iota ?x (result (action-term j action) ?x))

C:  (request    :sender i :receiver j
        :content (action j action))     ;
    (inform-ref :sender j :receiver i
        :content (iota ?x (result (action-term j action) ?x)))
```

The question then arises as to the differences between these actions. In summary, the meaning of the actions, are, respectively:

A:Agent i says to j "do *action*", but does not say anything about the result

B:Agent i says to j "tell me the result of doing *action*"

C:Agent i says to j "do *action*, and then inform me of the result of doing *action*".

In action B, the question can legitimately be asked whether the action is actually performed or not. It should be noted that *result* is a function in the domain language, SL in this case. Thus this question must really be devolved to the domain representation language. Some languages may be able to compute the meaning of an action without performing that action: this would be very useful for planning agents who may not wish to perform an action before considering its likely effects[11]. Other agents, such as expression simplifiers, do not want to be overburdened with the complexity of performing the simplification, then separately having to inform the questioner of the result of the simplification. Of course, if the meaning of the result predicate in a given context is that the action does, in fact, get done, then example C will likely result in the action being done twice.

Annex B
(normative/informative)

SL as a Content Language

This annex introduces a concrete syntax for the SL language that is compatible with the description in §8. This syntax, and its associated semantics, are suggested as a candidate *content language* for use in conjunction with FIPA ACL. In particular, the syntax is defined to be a sub-grammar of the very general s-expression syntax specified for message content in §6.4.

This content language is included in the specification on an *informative* basis. *It is not mandatory for any FIPA specification agent to implement the computational mechanisms necessary to process all of the constructs in this language.* However, SL is a general purpose representation formalism that may be suitable for use in a number of different agent domains.

Statement of conformance
The following definitions of SL, and subsets SL0, SL1 and SL2 are *normative defininitions* of these languages. That is, if a given agent chooses to implement a parser/interpreter for these languages, the following definitions must be adhered to. However, these languages are *informative suggestions* for the *use* of a content language: no agent is required as part of part 2 of this FIPA 97 specification to use the following content languages. However it should be noted that certain other parts of the FIPA 97 specification do make normative use of (some of) the following languages.

B.1 Grammar for SL concrete syntax

```
SLContentExpression       = SLWff
                          | SLIdentifyingExpression
                          | SLActionExpression.

SLWff                     = SLAtomicFormula
                          | "(" "not"            SLWff ")"
                          | "(" "and"            SLWff SLWff ")"
                          | "(" "or"             SLWff SLWff ")"
                          | "(" "implies"        SLWff SLWff ")"
                          | "(" "equiv"          SLWff SLWff ")"
                          | "(" SLQuantifier     SLVariable SLWff ")"
                          | "(" SLModalOp  SLAgent SLWff ")"
                          | "(" SLActionOp SLActionExpression ")".
                          | "(" SLActionOp
                                SLActionExpression SLWff ")".

SLAtomicFormula           = SLPropositionSymbol
                          | "(" "=" SLTerm SLTerm ")"
                          | "(" "result" SLTerm SLTerm ")"
                          | "(" SLPredicateSymbol SLTerm* ")"
                          | true
                          | false.

SLQuantifier              = "forall"
                          | "exists".

SLModalOp                 = "B"
                          | "U"
                          | "PG"
                          | "I".

SLActionOp                = "feasible"
                          | "done".
```

```
SLTerm                      = SLVariable
                            | SLConstant
                            | SLFunctionalTerm
                            | SLActionExpression
                            | SLIdentifyingExpression.

SLIdentifyingExpression     = "(" "iota" SLVariable SLWff ")"

SLFunctionalTerm            = "(" SLFunctionSymbol SLTerm* ")".

SLConstant                  = NumericalConstant
                            | Word
                            | StringLiteral.

NumericalConstant           = IntegerLiteral
                            | FloatingPointLiteral.

SLVariable                  = VariableIdentifier.

SLActionExpression          = "(" "action" SLAgent SLFunctionalTerm ")"
                            | ACLCommunicativeAct
                            | "(" "|" SLActionExpression SLActionExpression ")"
                            | "(" ";" SLActionExpression SLActionExpression ")".

SLPropositionSymbol         = Word.

SLPredicateSymbol           = Word.

SLFunctionSymbol            = Word.

SLAgent                     = AgentName.
```

B.1.1 Lexical definitions

```
Word                        = [~ "\0x00" - "\0x1f",
                               "(", ")", "#", "0"-"9", "-", "?"]
                              [~ "\0x00" - "\0x1f",
                               "(", ")"] *.

VariableIdentifier          = "?"
                              [~ "\0x00" - "\0x1f",
                               "(", ")"] *.

IntegerLiteral              = ( "-" )? DecimalLiteral
                            | ( "-" )? HexLiteral.

FloatingPointLiteral        = (["0"-"9"])+ "." (["0"-"9"])+ (Exponent)?
                            | (["0"-"9"])+ Exponent.

DecimalLiteral              = ["0"-"9"]+.

HexLiteral                  = "0" ["x", "X"] (["0"-"9","a"-"f","A"-"F"])+.

Exponent                    = ["e", "E"] (["+","-"])? (["0"-"9"])+.

StringLiteral               = "\""
                              ( [~ "\"" ] | "\\\"" )*
                              "\"".
```

B.2 Notes on SL content language semantics

This section contains explanatory notes on the intended semantics of the constructs introduced in §B.1 above.

B.2.1 Grammar entry point: SL content expression

An SL content expression may be used as the content of an ACL message. There are three cases:

— A proposition, which may be assigned a truth value in a given context. Precisely, it is a well-formed

formula using the rules described in SLWff. A proposition is used in the *inform* act, and other acts derived from it.

— An action, which can be performed. An action may be a single action, or a composite action built using the sequencing and alternative operators. An action is used as a content expression when the act is the *request* act, and other CA's derived from it.

— An identifying reference expression (IRE), which identifies an object in the domain. This is the iota operator, and is used in the *inform-ref* macro act and other acts derived from it.

B.2.2 SL Well-formed formula (SLWff)

A well-formed formula is constructed from an atomic formula, whose meaning will be determined by the semantics of the underlying domain representation, or recursively by applying one of the construction operators or logical connectives described in the grammar rule. These are:

— `(not <SLWff>)`
Negation. The truth value of this expresion is false if *SLWff* is true. Otherwise it is true.

— `(and <SLWff0> <SLWff1>)`
Conjunction. This expression is true iff well-formed formulae *SLWff0* and *SLWff1* are both true, otherwise it is false.

— `(or <SLWff0> <SLWff1>)`
Disjunction. This expression is false iff well-formed formulae *SLWff0* and *SLWff1* are both false, otherwise it is true.

— `(implies <SLWff0> <SLWff1>)`
Implication. This expression is true if either *SLWff0* is false, or alternatively if *SLWff0* is true and *SLWff1* is true. Otherwise it is false. The expression corresponds to the standard material implication connective: SLWff0 $\Rightarrow$ SLWff1.

— `(equiv <SLWff0> <SLWff1>)`
Equivalence. This expression is true if either *SLWff0* is true and *SLWff1* is true, or alternatively if *SLWff0* is false and *SLWff1* is false. Otherwise is is false.

— `(forall <variable> <SLWff>)`
Universal quantification. The quantifed expression is

Inter-agent Communication Language for Agents

One of the properties of agents is their ability to communicate with other agents. NFACT communicates with other agents to retrieve useful articles. The agent does not retrieve rules from other agents because the rules that are used need to be generated from the user's own preferences and not another user, is who may have different interests even within the same topic domain. An abstract language has been created to allow communication between other agents and the *Agent Coordinator*.

Daryl Moulder
1998-10-30

Disclaimer

Refine Search

Search Results -

Terms	Documents
(dialogue or speech) adj system and KQML	1

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L3

Refine Search

Recall Text | Clear | Interrupt

Search History

DATE: Wednesday, April 14, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=NO; OP=OR			
L3	(dialogue or speech) adj system and KQML	1	L3
L2	L1 and (dialogue or speech) adj system	1	L2
L1	706/12.ccls.	200	L1

END OF SEARCH HISTORY

Hit List

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 1 of 1 returned.

☐ 1. Document ID: US 6233570 B1

Using default format because multiple data bases are involved.

L2: Entry 1 of 1 File: USPT May 15, 2001

US-PAT-NO: 6233570
DOCUMENT-IDENTIFIER: US 6233570 B1

TITLE: Intelligent user assistance facility for a software program

DATE-ISSUED: May 15, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Horvitz; Eric	Kirkland	WA		
Breese; John S.	Mercer Island	WA		
Heckerman; David E.	Bellevue	WA		
Hobson; Samuel D.	Seattle	WA		
Hovel; David O.	Redmond	WA		
Klein; Adrian C.	Seattle	WA		
Rommelse; Jacobus A.	Westerhoven			NL
Shaw; Gregory L.	Kirkland	WA		

US-CL-CURRENT: 706/11; 706/12, 707/102

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw De

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Terms	Documents
L1 and (dialogue or speech) adj system	1

Display Format: - Change Format

Previous Page Next Page Go to Doc#

Hit List

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 1 of 1 returned.

☐ 1. Document ID: US 20030093419 A1

Using default format because multiple data bases are involved.

L3: Entry 1 of 1 File: PGPB May 15, 2003

PGPUB-DOCUMENT-NUMBER: 20030093419
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20030093419 A1

TITLE: System and method for querying information using a flexible multi-modal interface

PUBLICATION-DATE: May 15, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Bangalore, Srinivas	Morristown	NJ	US	
Johnston, Michael	Hoboken	NJ	US	
Walker, Marilyn A.	Morristown	NJ	US	
Whittaker, Stephen	Morristown	NJ	US	

US-CL-CURRENT: 707/3

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw De

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Terms	Documents
(dialogue or speech) adj system and KQML	1

Display Format: - Change Format

Previous Page Next Page Go to Doc#

NPL Services for Examiners

NEW ScienceDirect Journals

Wednesday, April 14, 2004

STIC's mission is to connect examiners to critical prior art by providing information services and access to NPL electronic resources and print collections. A STIC facility is located in each Technology Center.

Most of the electronic resources listed on these Web pages are accessed via the Internet. You must be authenticated for data to be accessed. ▶ Firewall Authentication

Specialized Information Resources for Technology Centers

Select a Technology Center

Technology Centers: GO

Information Resources and Services

- Breaking News on Emerging Technologies
- List of Major E-Resources
- List of eBook and eJournal Titles
- Reference Tools
- Legal Resources
- Nanotechnology
- STIC Online Catalog
- PLUS System
- Foreign Patent Services
- Translation Services
- Trademark Law Library

Request STIC Services from your Desktop

- Request a Prior Art Search
- Request Delivery of Article or Book Reference
- Request Purchase of a Book/Journal
- Request Foreign Patent Document
- Request a Translation

Request PLUS Search

Intranet Home | Index | Resources | Contacts | Internet | Search | Firewall | Web Services

Last Modified: 03/25/2004 13:36:15

TC2100: EIC Resources and Services

ScienceDirect Journals
Daily Breaking News on Emerging Technologies:
- Encryption
- Information & Data Security
- Internet Security

Wednesday, April 14, 2004

These resources and services provide examiners with access to critical prior art. Most of the electronic resources listed on these Web pages are accessed via the Internet. You must be authenticated for data to be accessed. ▶ Firewall Authentication

➪ indicates tools featured in TC's NPL training.

Information Resources

Information Resources by Class and Subclass

Databases

➪ ACM Digital Library
Business Source Corporate
(Multidisciplinary subject coverage)
Dialog Classic on the Web
(Training and password required.)
DTIC STINET
(Citations of Defense Technical Information Center scientific and technical documents)
EEDD Submission Form
Examiners' Electronic Digest Database (EEDD)
(Database of examiner submitted NPL)
EPOQUE
(EPO's databases, available on stand-alone terminal in CPK2, 4B40)

➪ IEEE Xplore
(Full page images of over 800,000 Electrical & Electronic Engineering articles, papers and standards, 1988 - present. Select content is available from 1952-1987.)
INSPEC
(Seven million well-indexed physics, EE, and IT abstracts, 1969-present)
IP.com
(Defensive disclosures published to the Disclosures IP.com database from various websites)
NTIS (National Technical Information Service)
(resource for government-funded scientific, technical, engineering, and business related information)
Proquest Direct
(Multidisciplinary subject coverage)
Readers' Guide to Periodical Literature
(citations to popular multidisciplinary magazines)
Research Disclosure
(Published monthly as a paper journal and now as an online database product with advanced full text searching capabilities for defensive disclosure information.)

Software Patent Institute (SPI) (Select "Free Access")
(Searchable database of Software Technologies.)
SPIE Digital Library
(journals and proceedings on optics and photonics)
STN on the Web (training and password required)
(The other link is via the Patent Examiner's Toolkit. On your computer, click on the START button, then on the PE Toolkit, then on STN Express.)
True Query
(A resurrected version of the old "Computer Select" database, providing full text access to over 100 technology focused publications, a glossary of technical terms, product reviews and over 60,000 product specifications from 1999 to the present. If html code appears on your screen, click browser's "Reload" or "Refresh" button.)

Books and Journals

Search STIC Online Catalog
InfoSECURITYnetBASE
(Information security)
Knovel
(Applied science and engineering)
NetLibrary.com
(Multidisciplinary subject coverage)
Safari Online Books
(Computer and information technology)
ScienceDirect
(scientific, technical, and medical journals)
Springer Publishing Company
(biotech, physics, and computer journals)

Daily Newspapers

Fulltext newspaper articles are available electronically in Proquest Direct.

CD-ROM Resources

Older full text NPL resources/articles received in CD-Rom format. These resources are available on EIC2100 PCs in CPK2, 4B40.

Equipment

Reference Tools

Bartleby.com
(Several versions of Roget's Thesaurus, a dictionary, an encyclopedia, quotations, English usage books and more.)
Computer References
(Dictionaries, Acronyms Finders, Encyclopedias)
Efunda
(30,000 pages of engineering fundamentals and calculators)
Encyclopedia Britannica
Encyclopedia of Software Engineering
Eric Weisstein's World of Mathematics
(A comprehensive online encyclopedia of mathematics.)
HowStuffWorks
(Search a term to find articles that explain how it works.)
Over 2000 Glossary Links
(Links to numerous technical, specialty, and general glossaries.)
PCWebopedia
Wiley Encyclopedia of Electrical and Electronics Engineering
Yourdictionary.com

(Numerous "specialty dictionaries"... technological, law, business related and more.)

Services

EIC2100 Staff
Foreign Patent Services
PLUS
Request a Book/Journal Purchase
Request a Book or Article
Request a Foreign Patent Publication
[e-submit] [Printable form]
Request a Prior Art Search
[e-submit] [Printable form]
Fast & Focused Search Criteria
STIC Online Catalog
Translation Services

Web Resources

A Brief History of the Hard Disk Drive

CiteSeer (ResearchIndex)
(Full text scientific research papers - in pdf and postscript formats.)

Internet Engineering Task Force
(The IETF Secretariat, run by The Corporation for National Research Initiatives with funding from the US government, maintains an index of Internet-Drafts.)

Nanotechnology

Requests for Comments (RFCs) Database
(Requests for Comments (RFC) document series is a set of technical and organizational notes about the Internet (originally the ARPANET), beginning in 1969 and discussing many aspects of computer networking, including protocols, procedures and concepts as well as meeting notes and opinions.)

Usenet Archive (Google Groups)

Wayback Machine
(Archived web pages.)

Submit comments and suggestions to Anne Hendrickson

To report technical problems, click here

IEEE Xplore® RELEASE 1.6

Welcome
United States Patent and Trademark Office

Over **1,024,576** documents available

Welcome to IEEE *Xplore*®

- Home
- What Can I Access?
- Log-out

Tables of Contents

- Journals & Magazines
- Conference Proceedings
- Standards

Search

- By Author
- Basic
- Advanced

Member Services

- Join IEEE
- Establish IEEE Web Account
- Access the IEEE Member Digital Library

IEEE ANNOUNCES NEW RELEASE FOR IEEE *XPLORE* ENHANCEMENTS - LEARN MORE.

IEEE *Xplore* provides full-text access to IEEE transactions, journals, magazines and conference proceedings published since 1988 plus select content back to 1950, and all current IEEE Standards.

FREE TO ALL: Browse tables of contents and access Abstract records of IEEE transactions, journals, magazines, conference proceedings and standards.

IEEE MEMBERS: Browse or search to access any complete Abstract record as well as articles from IEEE Spectrum Magazine. Access your personal online subscriptions using your active IEEE Web Account. If you do not have one, go to "Establish IEEE Web Account" to set up an account.

CORPORATE, GOVERNMENT AND UNIVERSITY SUBSCRIBERS: Search and access complete Abstract records and full-text documents of the IEEE online publications to which your institution subscribes.

IEEE Xplore® RELEASE 1.6

Welcome
United States Patent and Trademark Office

1) Enter a single keyword, phrase, or Boolean expression.
Example: acoustic imaging (means the phrase acoustic imaging plus any stem variations)
2) Limit your search by using search operators and field codes, if desired.
Example: optical <and> (fiber <or> fibre) <in> ti
3) Limit the results by selecting Search Options.
4) Click Search. See Search Examples

```
multiagent <or> multi-agent
<and> kqml
```

Start Search Clear

Note: This function returns plural and suffixed forms of the keyword(s).

Search operators: <and> <or> <not> <in> More

Field codes: au (author), ti (title), ab (abstract), jn (publication name), de (index term) More

Search Options:

Select publication types:

- [x] IEEE Journals
- [x] IEE Journals
- [x] IEEE Conference proceedings
- [x] IEE Conference proceedings
- [x] IEEE Standards

Select years to search:

From year: All to Present

Organize search results by:

Sort by: Relevance
In: Descending order

List 15 Results per page

Your search matched **5** of **1024576** documents.
A maximum of **500** results are displayed, **15** to a page, sorted by **Relevance Descending** order.

Refine This Search:
You may refine your search by editing the current search expression or enter new one in the text box.

multiagent <and> kqml Search

☐ Check to search within this result set

Results Key:
JNL = Journal or Magazine **CNF** = Conference **STD** = Standard

---

1 **Toward an open virtual market place for mobile agents**
Esmahi, L.; Dini, P.; Bernard, J.C.;
Enabling Technologies: Infrastructure for Collaborative Enterprises, 1999. (W ICE '99) Proceedings. IEEE 8th International Workshops on , 16-18 June 199
Pages:279 - 286

[Abstract] [PDF Full-Text (100 KB)] **IEEE CNF**

---

2 **A product retrieval system for electronic commerce based on KQML**
Jeong-Il Song; Han-Hyuk Chung; Eun-Seok Lee;
Parallel Processing, 1999. Proceedings. 1999 International Workshops on , 21 Sept. 1999
Pages:387 - 391

[Abstract] [PDF Full-Text (124 KB)] **IEEE CNF**

---

3 **Coordinating multiple agents in the supply chain**
Barbuceanu, M.; Fox, M.S.;
Enabling Technologies: Infrastructure for Collaborative Enterprises, 1996. Proceedings of the 5th Workshop on , 19-21 June 1996
Pages:134 - 141

[Abstract] [PDF Full-Text (1164 KB)] **IEEE CNF**

---

4 **Collaborative prototyping in distributed virtual reality using an agei communication language**
Nedelec, A.; Reignier, P.; Rodin, V.;
Systems, Man, and Cybernetics, 2000 IEEE International Conference on , Vol 2 , 8-11 Oct. 2000
Pages:1007 - 1012 vol.2

[Abstract] [PDF Full-Text (508 KB)] IEEE CNF

5 **Developing coherent multiagent systems using JAFMAS**
Chauhan, D.; Baker, A.D.;
Multi Agent Systems, 1998. Proceedings. International Conference on , 3-7 Jt
1998
Pages:407 - 408

[Abstract] [PDF Full-Text (16 KB)] IEEE CNF

Home | Log-out | Journals | Conference Proceedings | Standards | Search by Author | Basic Search | Advanced Search | Join IEEE | Web Account | New this week | OPAC Linking Information | Your Feedback | Technical Support | Email Alerting | No Robots Please | Release Notes | IEEE Online Publications | Help | FAQ | Terms | Back to Top

NPL Services for Examiners

NEW! ScienceDirect Journals

Wednesday, April 14, 2004

STIC's mission is to connect examiners to critical prior art by providing information services and access to NPL electronic resources and print collections. A STIC facility is located in each Technology Center.

Most of the electronic resources listed on these Web pages are accessed via the Internet. You must be authenticated for data to be accessed. ▶ Firewall Authentication.

Specialized Information Resources for Technology Centers

Select a Technology Center

Technology Centers: GO

Information Resources and Services

- Breaking News on Emerging Technologies
- List of Major E-Resources
- List of eBook and eJournal Titles
- Reference Tools
- Legal Resources
- Nanotechnology
- STIC Online Catalog
- PLUS System
- Foreign Patent Services
- Translation Services
- Trademark Law Library

Request STIC Services from your Desktop

- Request a Prior Art Search
- Request Delivery of Article or Book Reference
- Request Purchase of a Book/Journal
- Request Foreign Patent Document
- Request a Translation

Request PLUS Search

Intranet Home | Index | Resources | Contacts | Internet | Search | Firewall | Web Services

Last Modified: 03/25/2004 13:36:15

TC2100: EIC Resources and Services

ScienceDirect Journals
Daily Breaking News on Emerging Technologies:
- Encryption
- Information & Data Security
- Internet Security

Wednesday, April 14, 2004

These resources and services provide examiners with access to critical prior art. Most of the electronic resources listed on these Web pages are accessed via the Internet. You must be authenticated for data to be accessed. ▶ Firewall Authentication

➨ indicates tools featured in TC's NPL training.

Information Resources

Information Resources by Class and Subclass

Databases

➨ ACM Digital Library
Business Source Corporate
(Multidisciplinary subject coverage)
Dialog Classic on the Web
(Training and password required.)
DTIC STINET
(Citations of Defense Technical Information Center scientific and technical documents)
EEDD Submission Form
Examiners' Electronic Digest Database (EEDD)
(Database of examiner submitted NPL)
EPOQUE
(EPO's databases, available on stand-alone terminal in CPK2, 4B40)

➨ IEEE Xplore
(Full page images of over 800,000 Electrical & Electronic Engineering articles, papers and standards, 1988 - present. Select content is available from 1952-1987.)
INSPEC
(Seven million well-indexed physics, EE, and IT abstracts, 1969-present)
IP.com
(Defensive disclosures published to the Disclosures IP.com database from various websites)
NTIS (National Technical Information Service)
(resource for government-funded scientific, technical, engineering, and business related information)
Proquest Direct
(Multidisciplinary subject coverage)
Readers' Guide to Periodical Literature
(citations to popular multidisciplinary magazines)
Research Disclosure
(Published monthly as a paper journal and now as an online database product with advanced full text searching capabilities for defensive disclosure information.)

Software Patent Institute (SPI) (Select "Free Access")
(Searchable database of Software Technologies.)
SPIE Digital Library
(journals and proceedings on optics and photonics)
STN on the Web (training and password required)
(The other link is via the Patent Examiner's Toolkit. On your computer, click on the START button, then on the PE Toolkit, then on STN Express.)
True Query
(A resurrected version of the old "Computer Select" database, providing full text access to over 100 technology focused publications, a glossary of technical terms, product reviews and over 60,000 product specifications from 1999 to the present. If html code appears on your screen, click browser's "Reload" or "Refresh" button.)

Books and Journals

Search STIC Online Catalog
InfoSECURITYnetBASE
(Information security)
Knovel
(Applied science and engineering)
NetLibrary.com
(Multidisciplinary subject coverage)
Safari Online Books
(Computer and information technology)
ScienceDirect
(scientific, technical, and medical journals)
Springer Publishing Company
(biotech, physics, and computer journals)

Daily Newspapers

Fulltext newspaper articles are available electronically in Proquest Direct.

CD-ROM Resources

Older full text NPL resources/articles received in CD-Rom format. These resources are available on EIC2100 PCs in CPK2, 4B40.

Equipment

Reference Tools

Bartleby.com
(Several versions of Roget's Thesaurus, a dictionary, an encyclopedia, quotations, English usage books and more.)
Computer References
(Dictionaries, Acronyms Finders, Encyclopedias)
Efunda
(30,000 pages of engineering fundamentals and calculators)
Encyclopedia Britannica
Encyclopedia of Software Engineering
Eric Weisstein's World of Mathematics
(A comprehensive online encyclopedia of mathematics.)
HowStuffWorks
(Search a term to find articles that explain how it works.)
Over 2000 Glossary Links
(Links to numerous technical, specialty, and general glossaries.)
PCWebopedia
Wiley Encyclopedia of Electrical and Electronics Engineering
Yourdictionary.com

(Numerous "specialty dictionaries"... technological, law, business related and more.)

Services

EIC2100 Staff
Foreign Patent Services
PLUS
Request a Book/Journal Purchase
Request a Book or Article
Request a Foreign Patent Publication
[e-submit] [Printable form]
Request a Prior Art Search
[e-submit] [Printable form]
Fast & Focused Search Criteria
STIC Online Catalog
Translation Services

Web Resources

A Brief History of the Hard Disk Drive

CiteSeer (ResearchIndex)
(Full text scientific research papers - in pdf and postscript formats.)

Internet Engineering Task Force
(The IETF Secretariat, run by The Corporation for National Research Initiatives with funding from the US government, maintains an index of Internet-Drafts.)

Nanotechnology

Requests for Comments (RFCs) Database
(Requests for Comments (RFC) document series is a set of technical and organizational notes about the Internet (originally the ARPANET), beginning in 1969 and discussing many aspects of computer networking, including protocols, procedures and concepts as well as meeting notes and opinions.)

Usenet Archive (Google Groups)

Wayback Machine
(Archived web pages.)

Submit comments and suggestions to Anne Hendrickson

To report technical problems, click here

IEEE Xplore® RELEASE 1.6

Welcome
United States Patent and Trademark Office

Over **1,024,576** documents available

Welcome to IEEE *Xplore*®

- Home
- What Can I Access?
- Log-out

Tables of Contents

- Journals & Magazines
- Conference Proceedings
- Standards

Search

- By Author
- Basic
- Advanced

Member Services

- Join IEEE
- Establish IEEE Web Account
- Access the IEEE Member Digital Library

IEEE ANNOUNCES NEW RELEASE FOR IEEE *XPLORE* ENHANCEMENTS - LEARN MORE.

IEEE *Xplore* provides full-text access to IEEE transactions, journals, magazines and conference proceedings published since 1988 plus select content back to 1950, and all current IEEE Standards.

FREE TO ALL: Browse tables of contents and access Abstract records of IEEE transactions, journals, magazines, conference proceedings and standards.

IEEE MEMBERS: Browse or search to access any complete Abstract record as well as articles from IEEE Spectrum Magazine. Access your personal online subscriptions using your active IEEE Web Account. If you do not have one, go to "Establish IEEE Web Account" to set up an account.

CORPORATE, GOVERNMENT AND UNIVERSITY SUBSCRIBERS: Search and access complete Abstract records and full-text documents of the IEEE online publications to which your institution subscribes.

Your search matched **2** documents.

A maximum of **500** results are displayed, **15** to a page, sorted by **Relevance Descending** order.

Results Key:
JNL = Journal or Magazine **CNF** = Conference **STD** = Standard

1 **Tooling the lexicon acquisition process for large-scale KBMT**
Leavitt, J.R.R.; Lonsdale, D.W.; Keck, K.; Nyberg, E.H.;
Tools with Artificial Intelligence, 1994. Proceedings., Sixth International Conf on , 6-9 Nov. 1994
Pages:283 - 289

[Abstract] [PDF Full-Text (648KB)] **IEEE CNF**

2 **The DIBBS blackboard control architecture and its application to distributed natural language processing**
Leavitt, J.R.R.; Nyberg, E.;
Tools for Artificial Intelligence, 1990.,Proceedings of the 2nd International IE Conference on , 6-9 Nov. 1990
Pages:202 - 208

[Abstract] [PDF Full-Text (664KB)] **IEEE CNF**